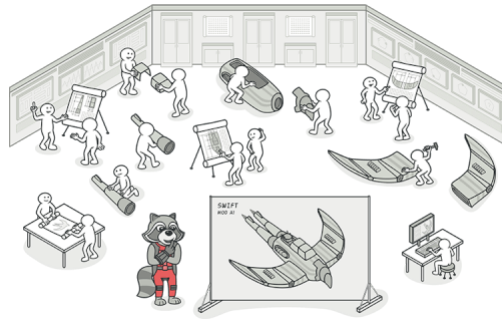




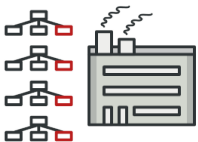
Ayuda a Ucrania a detener a Rusia



PATRONES DE DISEÑO

El catálogo de ejemplos en Swift

Patrones creacionales



Abstract Factory ★★★

Permite producir familias de objetos relacionados sin especificar sus clases concretas.

 [Artículo principal](#)

 [Uso en Swift](#)

 [Ejemplo conceptual](#)

 [Ejemplo del mundo real](#)

Builder ★★★



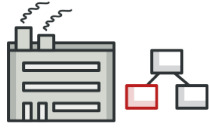
Permite construir objetos complejos paso a paso. Este patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.

 [Artículo principal](#)

 [Ejemplo conceptual](#)

 [Uso en Swift](#)

 [Ejemplo del mundo real](#)



Factory Method ★★★

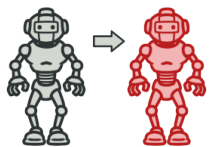
Proporciona una interfaz para la creación de objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.

 [Artículo principal](#)

 [Ejemplo conceptual](#)

 [Uso en Swift](#)

 [Ejemplo del mundo real](#)



Prototype ★★☆☆

Permite copiar objetos existentes sin que el código dependa de sus clases.

 [Artículo principal](#)

 [Ejemplo conceptual](#)

 [Uso en Swift](#)

 [Ejemplo del mundo real](#)



Singleton ★★☆☆

Permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.

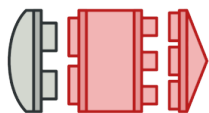
 [Artículo principal](#)

 [Ejemplo conceptual](#)

 [Uso en Swift](#)

 [Ejemplo del mundo real](#)

Patrones estructurales



Adapter ★★★

Permite la colaboración entre objetos con interfaces incompatibles.

 [Artículo principal](#)

 [Ejemplo conceptual](#)

 [Uso en Swift](#)

 [Ejemplo del mundo real](#)

Bridge ★☆☆☆

Permite dividir una clase grande o un grupo de clases estrechamente relacionadas, en dos jerarquías separadas (abstracción e implementación) que pueden desarrollarse



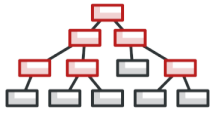
independientemente la una de la otra.

 [Artículo principal](#)

 [Uso en Swift](#)

 [Ejemplo conceptual](#)

 [Ejemplo del mundo real](#)



Composite ★★★

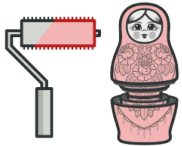
Permite componer objetos en estructuras de árbol y trabajar con esas estructuras como si fueran objetos individuales.

 [Artículo principal](#)

 [Uso en Swift](#)

 [Ejemplo conceptual](#)

 [Ejemplo del mundo real](#)



Decorator ★★★

Permite añadir funcionalidades a objetos colocando estos objetos dentro de objetos encapsuladores especiales que contienen estas funcionalidades.

 [Artículo principal](#)

 [Uso en Swift](#)

 [Ejemplo conceptual](#)

 [Ejemplo del mundo real](#)



Facade ★★★

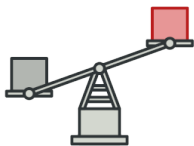
Proporciona una interfaz simplificada a una biblioteca, un framework o cualquier otro grupo complejo de clases.

 [Artículo principal](#)

 [Uso en Swift](#)

 [Ejemplo conceptual](#)

 [Ejemplo del mundo real](#)



Flyweight ★☆☆

Permite mantener más objetos dentro de la cantidad disponible de memoria RAM compartiendo las partes comunes del estado entre varios objetos en lugar de mantener toda la información en cada objeto.

 [Artículo principal](#)

 [Uso en Swift](#)

 [Ejemplo conceptual](#)

 [Ejemplo del mundo real](#)



Proxy ★☆☆

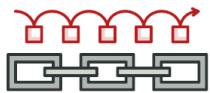
Permite proporcionar un sustituto o marcador de posición para otro objeto. Un proxy controla el acceso al objeto original, permitiéndote hacer algo antes o después de que la solicitud llegue al objeto original.

 [Artículo principal](#)

 [Ejemplo conceptual](#)

 [Uso en Swift](#) [Ejemplo del mundo real](#)

Patrones de comportamiento



Chain of Responsibility ★☆☆

Permite pasar solicitudes a lo largo de una cadena de manejadores. Al recibir una solicitud, cada manejador decide si la procesa o si la pasa al siguiente manejador de la cadena.

 [Artículo principal](#) [Ejemplo conceptual](#) [Uso en Swift](#) [Ejemplo del mundo real](#)

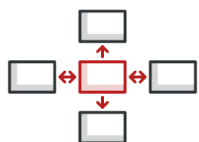
Command ★★★

Convierte una solicitud en un objeto independiente que contiene toda la información sobre la solicitud. Esta transformación te permite parametrizar los métodos con diferentes solicitudes, retrasar o poner en cola la ejecución de una solicitud y soportar operaciones que no se pueden realizar.

 [Artículo principal](#) [Ejemplo conceptual](#) [Uso en Swift](#) [Ejemplo del mundo real](#)

Iterator ★★★

Permite recorrer elementos de una colección sin exponer su representación subyacente (lista, pila, árbol, etc.).

 [Artículo principal](#) [Ejemplo conceptual](#) [Uso en Swift](#) [Ejemplo del mundo real](#)

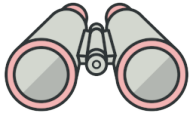
Mediator ★★☆☆

Permite reducir las dependencias caóticas entre objetos. El patrón restringe las comunicaciones directas entre los objetos, forzándolos a colaborar únicamente a través de un objeto mediador.

 [Artículo principal](#) [Ejemplo conceptual](#) [Uso en Swift](#) [Ejemplo del mundo real](#)

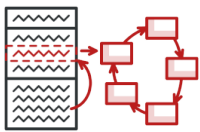
Memento ★☆☆

Permite guardar y restaurar el estado previo de un objeto sin revelar los detalles de su implementación.

[!\[\]\(2e897e890e69d81eae4503a8342c36b0_img.jpg\) Artículo principal](#)[!\[\]\(bd1a142de767a21e5362c595f844a4ff_img.jpg\) Uso en Swift](#)[!\[\]\(e2376d476d06eb31946dc01a69a4403a_img.jpg\) Ejemplo conceptual](#)[!\[\]\(74d4806277d7e73349d8e8c0897931e9_img.jpg\) Ejemplo del mundo real](#)

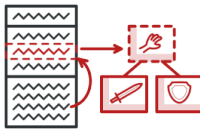
Observer ★★★

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

[!\[\]\(8bba887393ca45b761e5cb49e755e762_img.jpg\) Artículo principal](#)[!\[\]\(6bb0e4f14c4133b37d2887cb37e67ddd_img.jpg\) Uso en Swift](#)[!\[\]\(47734e4656765d20df4fdbd5b7aff048_img.jpg\) Ejemplo conceptual](#)[!\[\]\(bd3b31712ad9bab5a241210fa6925cdd_img.jpg\) Ejemplo del mundo real](#)

State ★★☆☆

Permite a un objeto alterar su comportamiento cuando su estado interno cambia. Parece como si el objeto cambiara su clase.

[!\[\]\(7bc43b319a082987e20f7bf78f4bab80_img.jpg\) Artículo principal](#)[!\[\]\(e50091943b385fe16d3277389202856f_img.jpg\) Uso en Swift](#)[!\[\]\(4436e6b00b9d5e62c2a161129eb3e4d0_img.jpg\) Ejemplo conceptual](#)[!\[\]\(179f167ede0522ebb4ea025b3ad78ca7_img.jpg\) Ejemplo del mundo real](#)

Strategy ★★★

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


[!\[\]\(5ddb2a112276baa148775929432349f9_img.jpg\) Artículo principal](#)[!\[\]\(fa03f7688acce2280e23104ced18e610_img.jpg\) Uso en Swift](#)[!\[\]\(fb9e809951d718d0a8038dca8a708d54_img.jpg\) Ejemplo conceptual](#)[!\[\]\(008bfeb2de157dcb66edb3a8218c280e_img.jpg\) Ejemplo del mundo real](#)

Template Method ★★☆☆

Define el esqueleto de un algoritmo en la superclase pero permite que las subclasses sobrescriban pasos del algoritmo sin cambiar su estructura.

[!\[\]\(141489a9a09a5a55d166fd7134726d50_img.jpg\) Artículo principal](#)[!\[\]\(f3cd43c0876202a7cb76d17dba19e77d_img.jpg\) Ejemplo conceptual](#)[Inicio](#)[Refactorización](#)[Patrones de diseño](#)[Contenido Premium](#)[Foro](#)[Contáctanos](#)

© 2014-2022 Refactoring.Guru. Todos los derechos reservados

 Ilustraciones por Dmitry Zhart

 Khmelnytske shosse 19 / 27, Kamianets-Podilskyi, Ucrania, 32305

 Email: support@refactoring.guru

[Términos y condiciones](#)

[Política de privacidad](#)

[Política de uso de contenido](#)