PATRONES DE DISEÑO

OBJETIVOS

- Comprender qué es un Patrón de Diseño.
- Comprender la utilidad de los Patrones de Diseño.
- Conocer cuáles son los patrones de diseño creacionales
- Identificar cuándo aplicar cada uno de los patrones de diseño creacionales.

¿QUÉ ES UN PATRÓN?

En el contexto de diseño de software ...

1

INTRODUCCIÓN

Contextualizando ...

LOS PROBLEMAS A RESOLVER ...

- Durante el diseño uno se encuentra con problemas (y soluciones) previamente abordados en otros proyectos.

- Diseñadores expertos saben que no deben resolver cada problema desde cero; sino, deben reutilizar buenas soluciones que funcionaron en el pasado.

2

DEFINICIONES

¿Qué es un Patrón de Diseño?

> Describe un problema de diseño recurrente a ser resuelto, la solución al mismo y el contexto bajo el que debe de utilizarse.

(Gamma et al. 1995; Buschmann et al. 1996)

¿QUÉ SON LOS PATRONES DE DISEÑO?

- Descripciones de objetos y clases que se comunican (relaciones) que han sido personalizados para resolver un problema de diseño general en un contexto particular.
- 'Alias' microarquitectura.
- Ayudan a los programadores en la optimización de su tiempo.

ELEMENTOS DE LOS PATRONES DE DISEÑO

- Nombre del Patrón: Identificador del patrón.
- Problema: ¿Cuándo aplicar el patrón?
- Solución: Describe el diseño, las relaciones y responsabilidades.
- Consecuencias: Resultados, consideraciones e implicancias del uso del patrón.

3

ORGANIZACIÓN

Clasificación de los patrones ...

ORGANIZACIÓN DE LOS PATRONES

- Análisis: Aplican a problemas de conceptualización.
- Arquitectura: Mayor tamaño que los de Diseño. Describen las estructuras de subsistemas completos.
- Diseño: Grupo pequeño de objetos interrelacionados.
- Codificación: Conocidos como idiomas. Son patrones de diseño específicos a un lenguaje de programación.

CLASIFICACIÓN DE LOS PATRONES DE DISEÑO

		Propósito		
		Creacional	**Estructural**	**De comportamiento**
Alcance	**Clase**	Factory Method	Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight (195) Observer State Strategy Visitor

DESCRIPCIÓN SEGÚN ALCANCE

Clases

- Se enfocan en las relaciones entre clases padres e hijas.
- Las relaciones de tipo herencia son estáticas (fijas al compilar).

Objetos

- Se enfocan en las relaciones entre objetos (en tiempo de ejecución).
- Relaciones dinámicas que pueden cambiar en tiempo de ejecución.

DESCRIPCIÓN SEGÚN PROPÓSITO

Creacionales

- Creación de objetos.
- Flexibilizar el código.
- Reutilizar el código.

Estructurales

- Organizan clases y objetos en estructuras.
- Estructuras flexibles.
- Estructuras eficientes.

De comportamiento

- Relacionado a algoritmos.
- Responsabilidades entre objetos.

4

PATRONES DE DISEÑO CREACIONALES

Detallemos algunos de los patrones ...

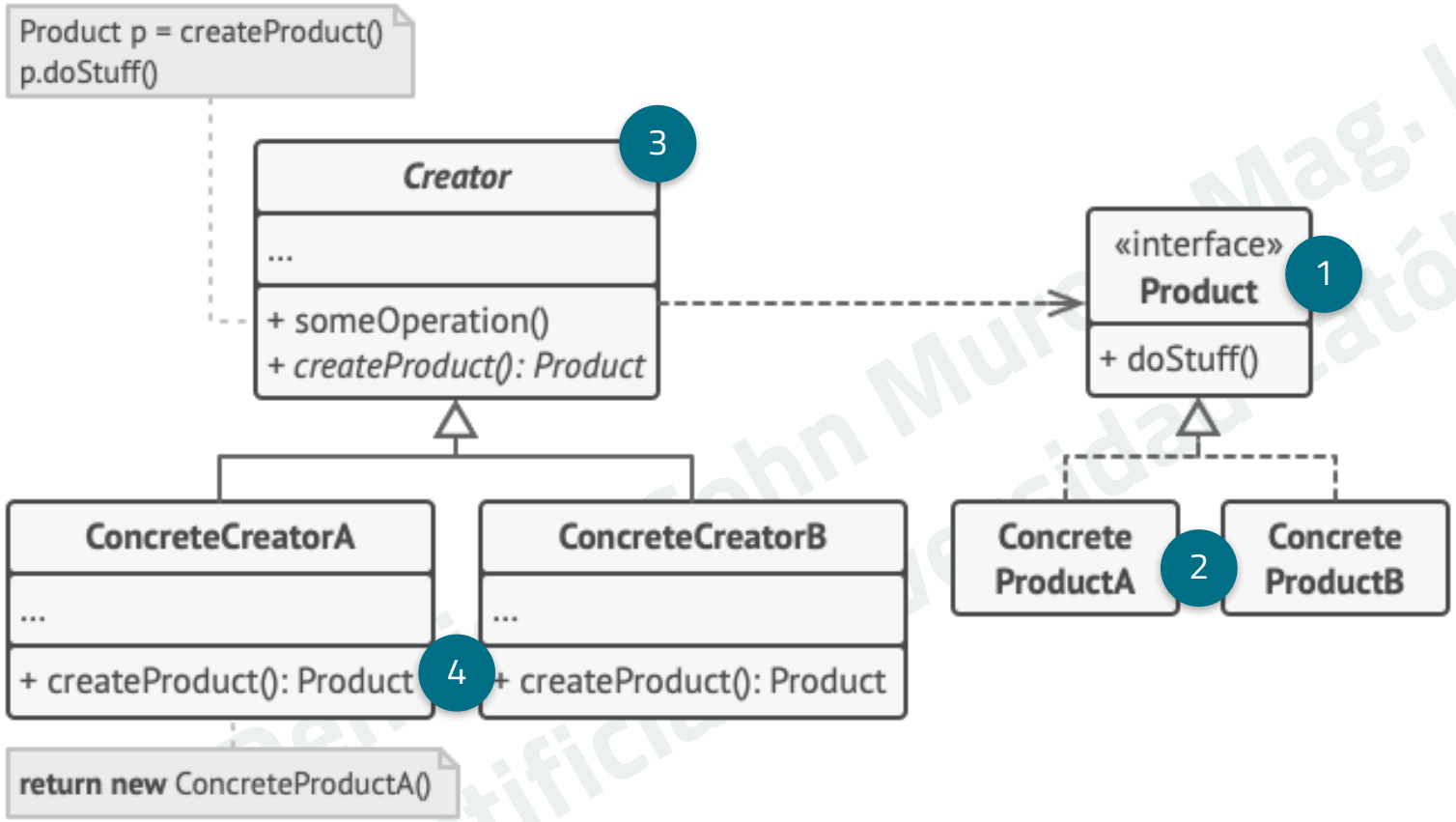
PATRÓN: FACTORY METHOD

PROBLEMA QUE CUBRE

- Los *frameworks* necesitan estandarizar el modelo de la arquitectura para todas las aplicaciones; pero deben permitir la definición de objetos para dominios particulares.

¿CÓMO LO HACE?

Define una interfaz para crear un objeto; pero las subclases deciden que clase se instanciará.



1. Interfaz de Producto: común a todos los objetos que puede producir.
2. Productos concretos: implementaciones de la interfaz Producto.
3. Clase Creadora Abstracta: declara el método que devuelve nuevos objetos de producto.
4. Creadores Concretos: devuelven un tipo particular de producto.

CONSECUENCIAS: FACTORY METHOD

- (+) Separa el código de creación del producto del código que lo utiliza. Esto facilita la extensibilidad de nuevos productos.
- (+) El Factory Method puede optar por devolver un objeto existente en lugar de crear uno nuevo.

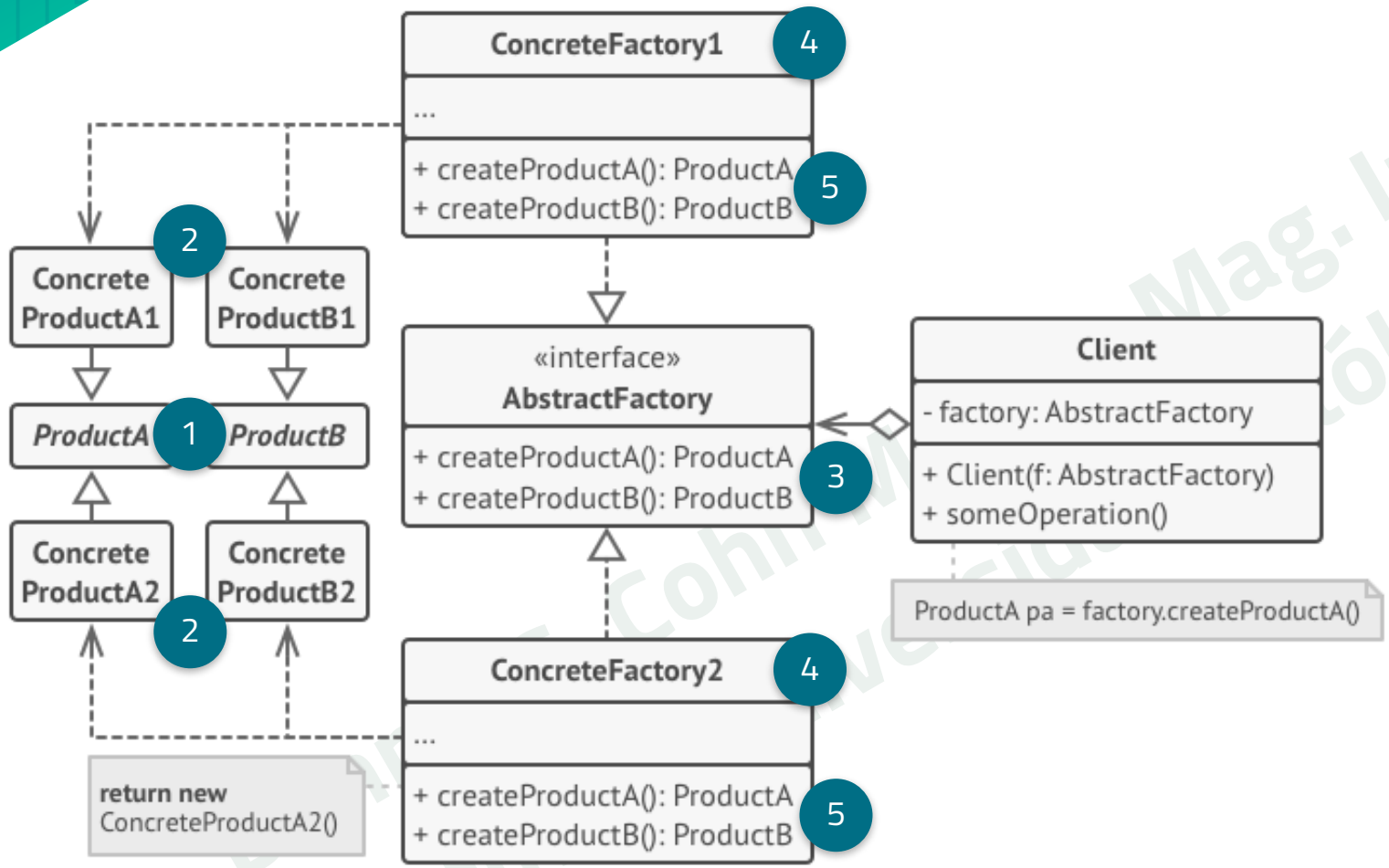
PATRÓN: ABSTRACT FACTORY

PROBLEMA QUE CUBRE

- Garantizar extensibilidad y portabilidad de la solución.
- Evita el uso de condiciones anidadas para instanciar clases.

¿CÓMO LO HACE?

Permite aprovisionar una familia de objetos relacionados sin necesidad de especificar sus clases concretas.



1. Clase Abstracta por tipo de Productos.
2. Implementar colecciones de Productos por cada tipo.
3. Interfaz de la Factoría que declara los métodos para instanciar cada tipo de producto.
4. Implementación de la Factoría por cada colección.
5. Es importante que se retorne la clase abstracta correspondiente.

CONSECUENCIAS: ABSTRACT FACTORY

- (+) Encapsula las clases concretas; limita los tipos de objetos que una aplicación crea.
- (+) Facilita el intercambio de las familias de productos; con solo el intercambio del ConcreteFactory.
- (+) Promueve consistencia entre productos a través de la creación de una familia de productos.
- (-) Es complejo agregar nuevos tipos de productos; ello implica modificar el AbstractFactory y sus clases hijas.

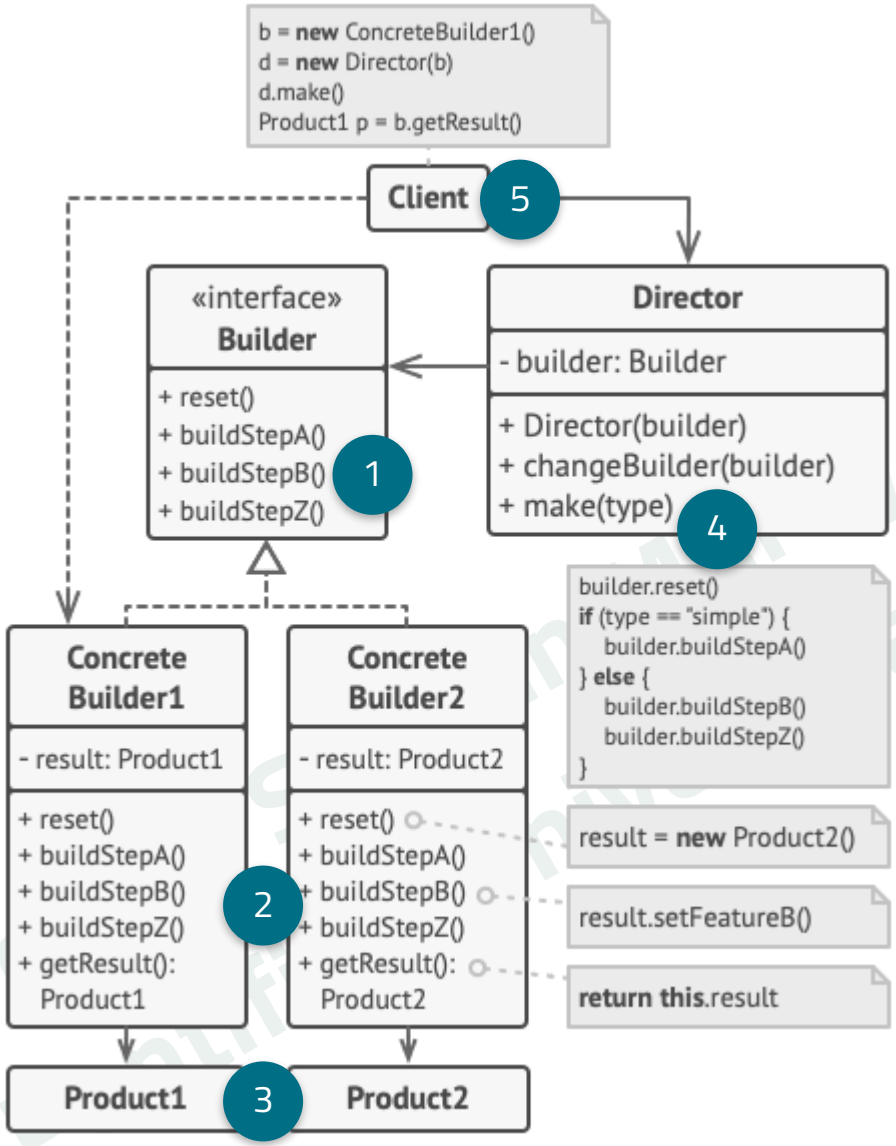
PATRÓN: BUILDER

PROBLEMA QUE CUBRE

- Cómo construir objetos complejos paso a paso.

¿CÓMO LO HACE?

Organiza la construcción de objetos en una serie de pasos. Para crear un objeto, se ejecuta una serie de estos pasos desde un objeto constructor.



1. Interfaz Constructora declara pasos de construcción comunes.
2. Constructores Concretos implementan los pasos de construcción.
3. Los Productos son los objetos resultantes. No heredan de los Constructores
4. Clase Directora define el orden de los pasos de construcción
5. El Cliente define el Constructor a utilizar.

CONSECUENCIAS: BUILDER

- (+) Abstrae la estructura del producto y los pasos que se siguen para su ensamblado.
- (+) Control granular sobre los pasos a seguir para el proceso de construcción.
- (+) Evita sobrecargar o complejizar el método constructor.

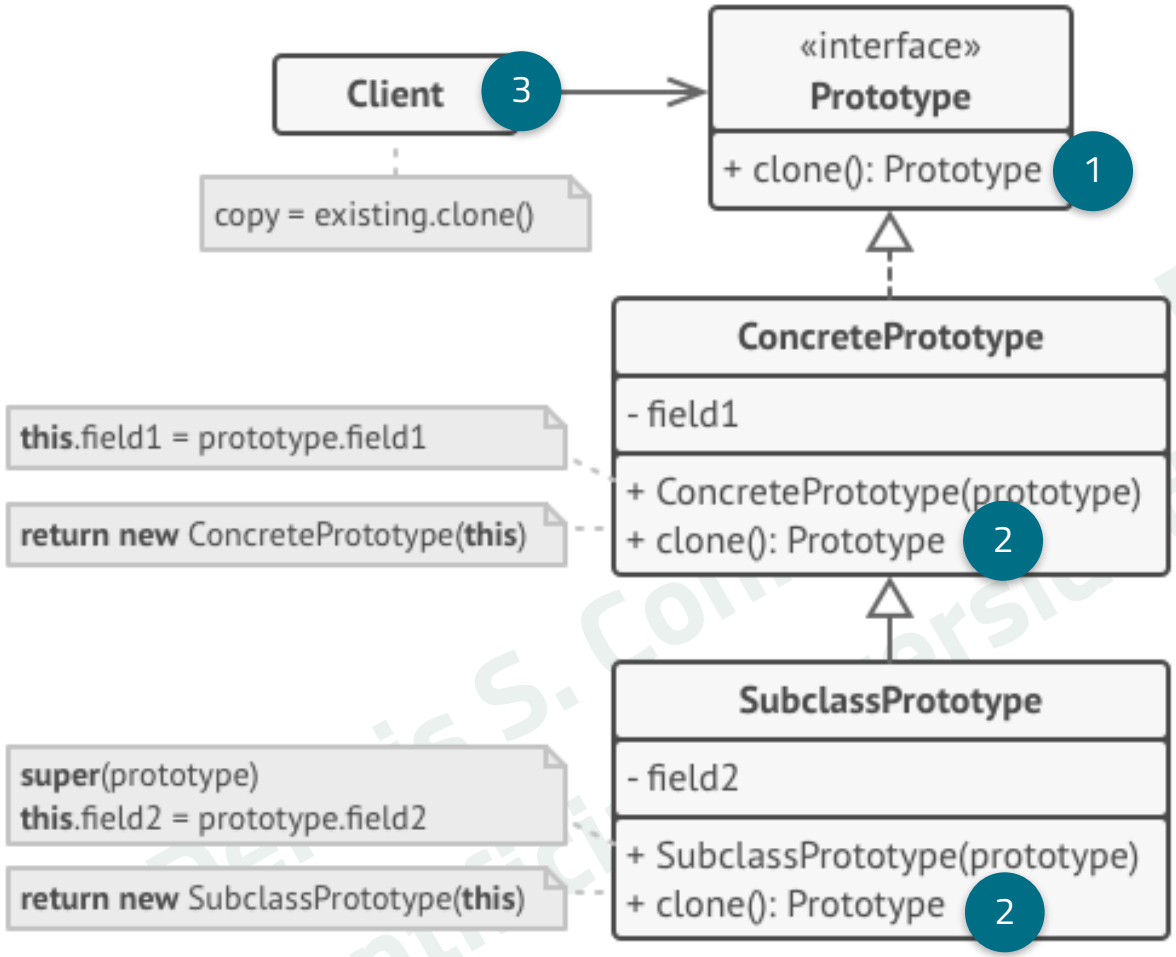
PATRÓN: PROTOTYPE

PROBLEMA QUE CUBRE

- Copiar objetos existentes sin conocer la estructura interna del objeto.

¿CÓMO LO HACE?

Delega el proceso de clonación a los propios objetos que están siendo clonados. El patrón declara una interfaz común para todos los objetos que soportan la clonación.



1. Interfaz Prototipo declara los métodos de clonación.

2. Prototipo Concreto implementa el método de clonación. Copia la información del objeto original al clon.

3. Cliente puede producir una copia de cualquier objeto que siga la interfaz del prototipo.

CONSECUENCIAS: PROTOTYPE

- (+) Permite incorporar una nueva clase concreta con solo registrar una nueva instancia.
- (+) No se requiere una clase Factoría. Se pueden instanciar nuevos objetos en base a la clonación.
- (-) Complejidad al clonar objetos con referencias circulares.

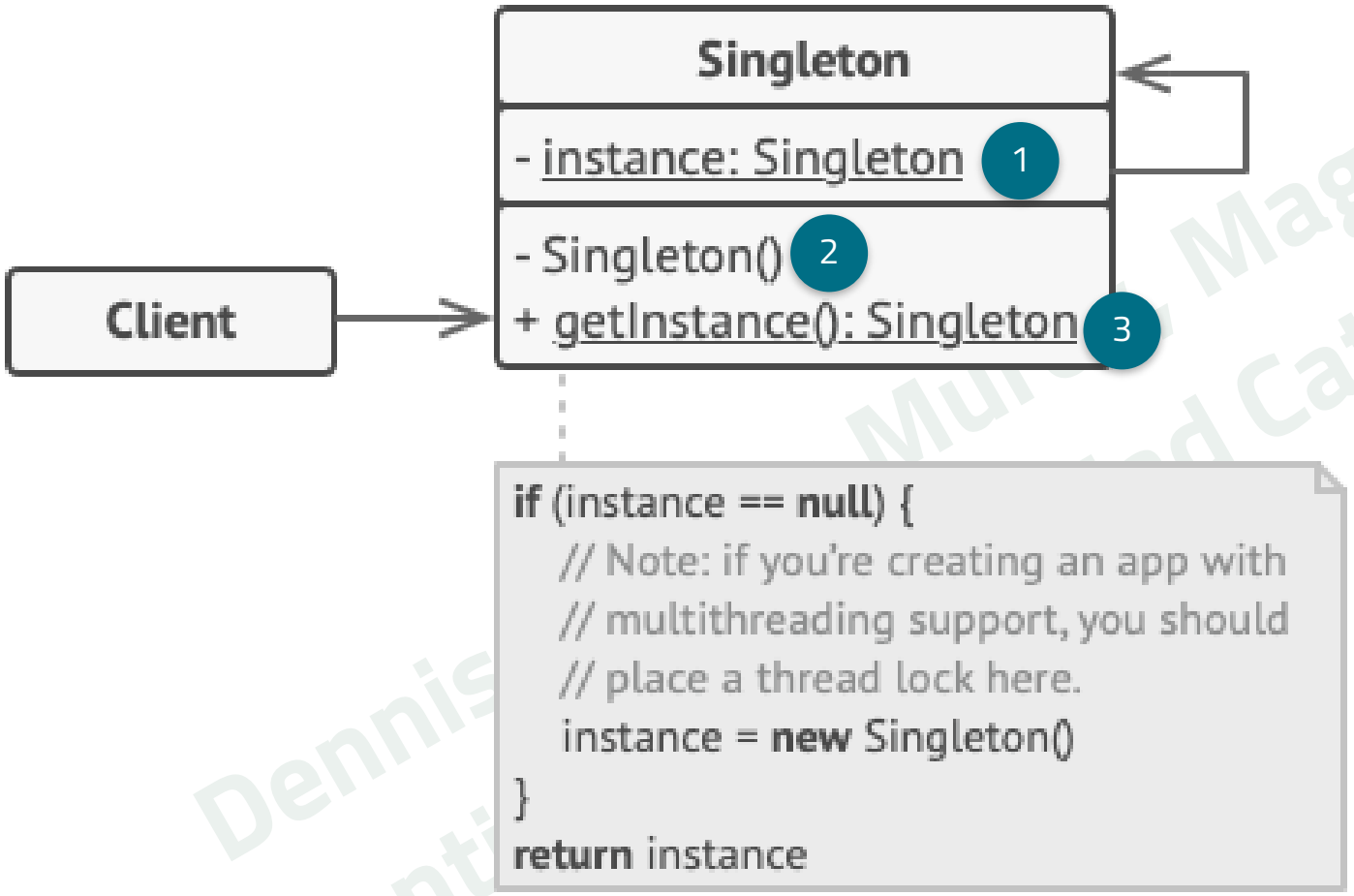
PATRÓN: SINGLETON

PROBLEMA QUE CUBRE

- Punto de acceso único para un recurso.
- Acceso global inmutable.

¿CÓMO LO HACE?

Asegura que una clase tiene una única instancia.



1. Instancia estática y privada de la misma clase.
2. Constructor privado
3. Método estático y público para obtener acceso a la instancia

CONSECUENCIAS: SINGLETON

- (+) Acceso controlado a una sola instancia.
- (+) Puede ajustarse para soportar un número máximo de instancias por objeto.
- (-) Puede enmascarar un mal diseño.

5

REFERENCIAS

BIBLIOGRAFÍA

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design patterns: elements of reusable object-oriented software. Pearson Deutschland GmbH.
- Freeman, E., & Robson, E. (2020). Head First Design Patterns. O'Reilly Media.
- Design patterns and refactoring. (n.d.). https://sourcemaking.com/design_patterns

Créditos:

- Plantilla de la presentación por SlidesCarnival
- Fotografías por Unsplash
- Diseño del fondo Hero Patterns