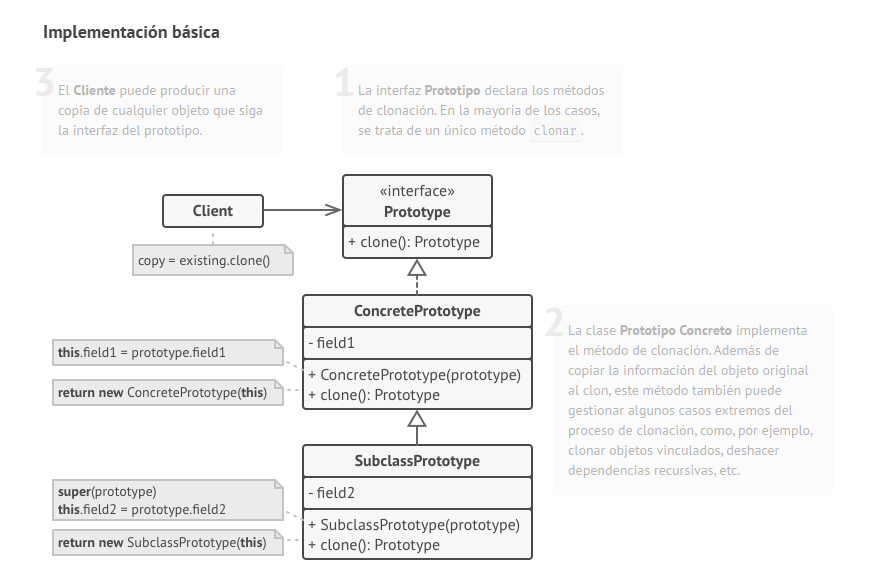
**PROTOTYPE**

****

* 1. **¿Qué es el patrón de diseño Prototype?**  
     El patrón Prototype es un patrón creacional que permite crear nuevos objetos copiando instancias existentes, conocidas como prototipos. En lugar de crear un objeto desde cero usando el operador new, se crea una copia (clon) de un objeto prototipo. Esto es útil cuando la creación de un objeto es costosa o compleja, o cuando se necesita conservar el estado de un objeto base. El patrón facilita la duplicación de objetos sin acoplar el código al tipo concreto de los objetos a clonar.
  2. **¿Cuál es el objetivo del patrón Prototype?**  
     Su objetivo es permitir la creación eficiente de nuevos objetos mediante la clonación de otros objetos ya configurados, en lugar de crear cada instancia manualmente. Esto se traduce en mayor flexibilidad, ya que el sistema puede generar nuevos objetos sin conocer sus clases exactas, y también mejora el rendimiento cuando los objetos tienen costos altos de inicialización (por ejemplo, configuraciones, conexiones o cálculos intensivos).
  3. **¿Qué problema resuelve el patrón Prototype?**  
     Resuelve el problema de crear múltiples objetos similares sin necesidad de repetir la lógica de construcción. En escenarios donde los objetos son costosos de crear, altamente configurables o tienen múltiples estados posibles, el patrón permite duplicarlos sin recrearlos desde cero. También ayuda cuando el sistema debe operar con objetos cuyo tipo exacto no se conoce hasta tiempo de ejecución.

1. **¿En qué situaciones es preferible usar Prototype en lugar de new?**  
   Es preferible usar Prototype cuando:

* La creación de objetos es muy costosa (por ejemplo, contiene muchos cálculos o requiere recursos).
* Se necesita crear múltiples instancias similares con pequeñas variaciones.
* El tipo exacto del objeto no se conoce hasta tiempo de ejecución.
* Se trabaja con configuraciones dinámicas que deben duplicarse.  
  Por ejemplo, en videojuegos donde se generan enemigos o personajes con atributos similares, se puede clonar un prototipo en lugar de reconstruirlos desde cero.

**¿Qué se necesita para que un objeto sea clonable?**  
El objeto debe implementar una interfaz o método que defina la operación de clonación, como clone() en Java o implementar el módulo copy en Python. Es importante que el método clone tenga acceso a todos los atributos del objeto y se asegure de copiar correctamente tanto los valores primitivos como las referencias a otros objetos (en caso de necesitar una copia profunda). En muchos lenguajes también se requiere que las clases sean marcadas como clonables (por ejemplo, Cloneable en Java).

**¿Cuál es la diferencia entre una copia superficial y una copia profunda en Prototype?**

* **Copia superficial (shallow copy):** copia el objeto pero no sus objetos internos. Las referencias dentro del objeto apuntan a las mismas instancias que el original.
* **Copia profunda (deep copy):** copia el objeto y también todos los objetos que contiene recursivamente, creando una duplicación completa sin compartir referencias.  
  Usar copia superficial puede ser más rápido, pero puede causar errores si se modifica un objeto que es compartido. Una copia profunda ofrece mayor independencia entre objetos clonados.

**Menciona una ventaja del patrón Prototype.**  
Una gran ventaja es la eficiencia en la creación de objetos. En lugar de pasar por todo el proceso de configuración o construcción, se puede copiar un prototipo ya configurado. Esto ahorra tiempo y recursos, especialmente en sistemas donde se crean muchos objetos similares. Además, reduce la dependencia del sistema de las clases concretas, promoviendo un diseño más flexible y extensible.

1. **¿Cómo se implementa el patrón Prototype en Java o C#?**  
   En **Java**, se implementa típicamente haciendo que la clase implemente la interfaz Cloneable y sobreescriba el método clone():

java

CopiarEditar

public class Producto implements Cloneable {

private String nombre;

public Producto clone() throws CloneNotSupportedException {

return (Producto) super.clone();

}

}

En **C#**, se implementa con la interfaz ICloneable:

csharp

CopiarEditar

public class Producto : ICloneable {

public string Nombre { get; set; }

public object Clone() {

return this.MemberwiseClone(); // Copia superficial

}

}

1. **¿Qué riesgos pueden surgir al usar el patrón Prototype?**  
   Los riesgos incluyen:

* Problemas al usar copias superficiales si los objetos tienen referencias compartidas que deberían ser independientes.
* Dificultad para mantener la lógica de clonación en clases con estructuras complejas.
* Posibles errores si el clon no copia todos los atributos correctamente.  
  Además, no todos los lenguajes manejan bien la clonación por defecto, por lo que a veces es necesario implementar lógicas de clonación personalizadas.

1. **Proporciona un ejemplo práctico del uso del patrón Prototype.**  
   Supongamos que estás desarrollando una aplicación para diseñar formularios. Cada tipo de formulario (registro, encuesta, feedback) tiene una estructura similar, pero con pequeños cambios. En lugar de construir cada formulario desde cero, puedes tener un prototipo base con campos comunes, y luego clonar este prototipo para crear formularios personalizados agregando o removiendo campos según necesidad. Esto reduce tiempo de configuración y asegura consistencia estructural.