Patrones de Creación: Builder, Singleton y Prototype

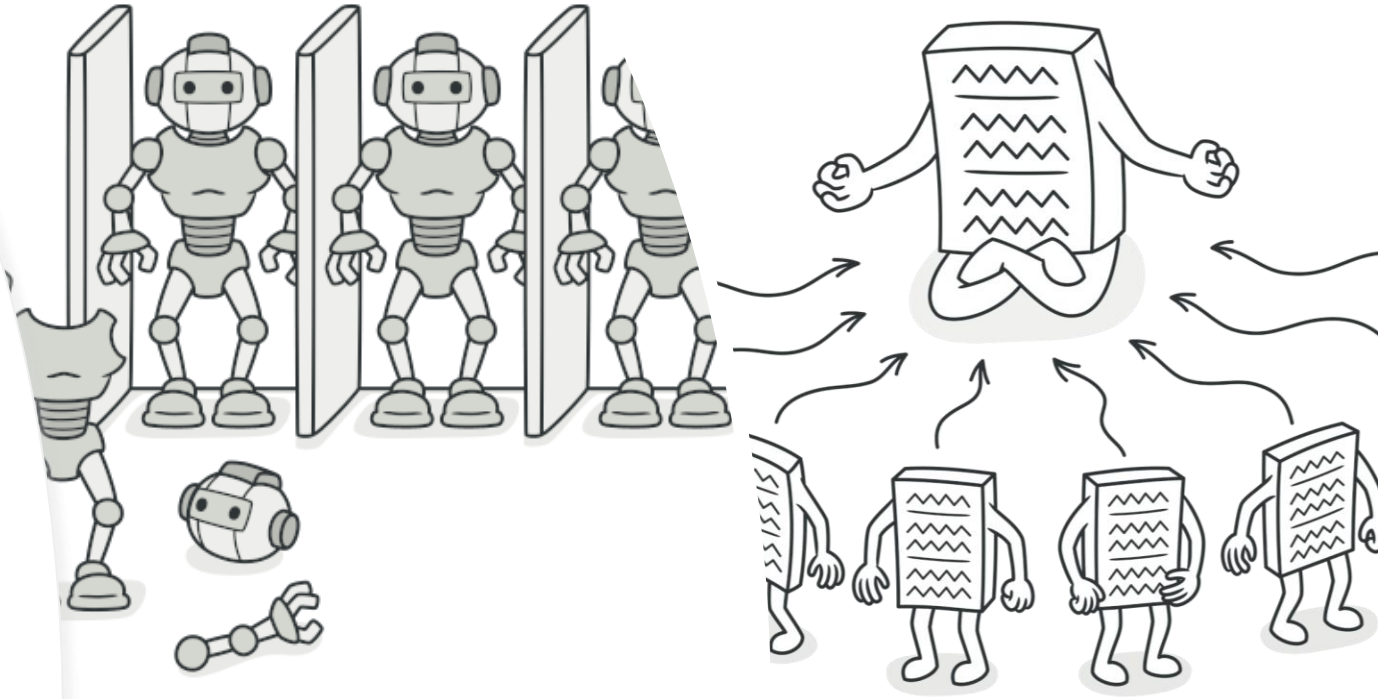
Estudiantes:

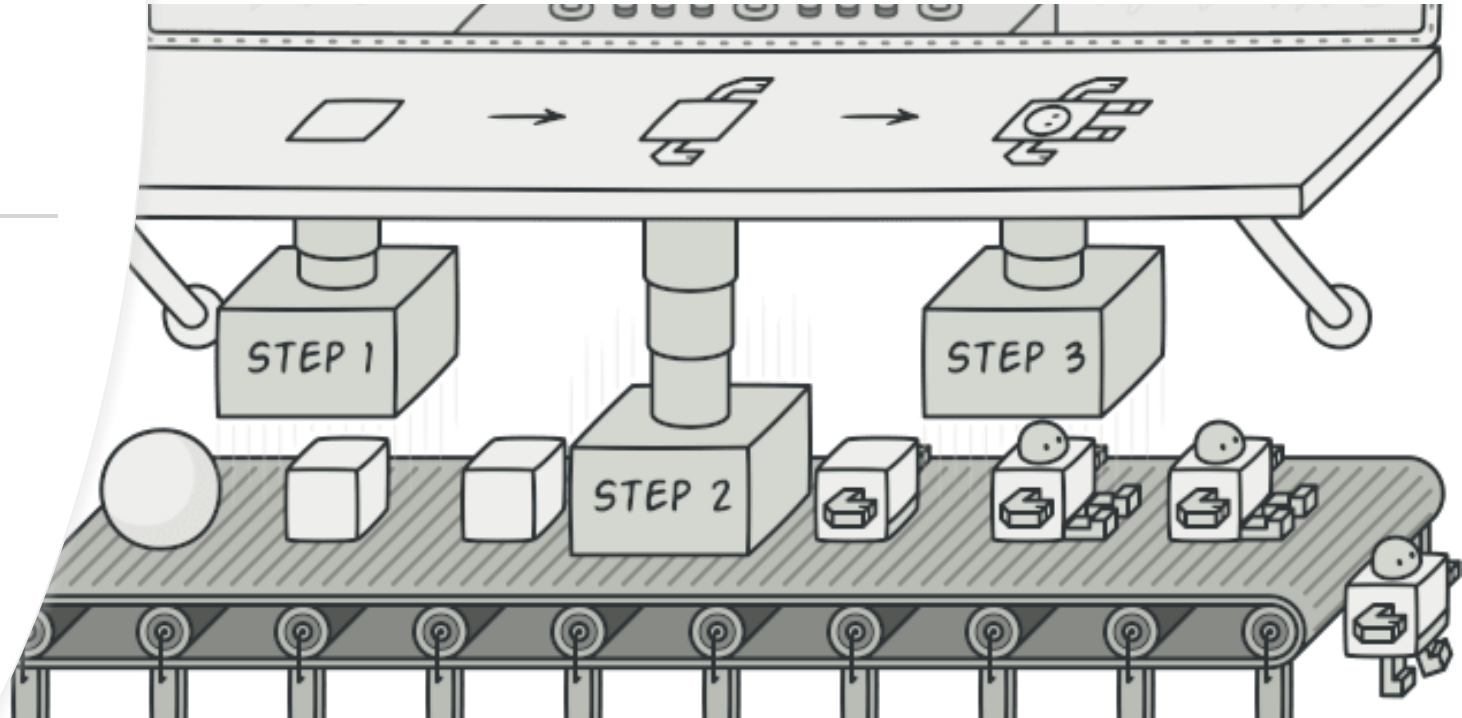
Luis Mejía 8-949-350

Pablo Paladino E-8-165038

Alejandra Caballero 8-943-1867

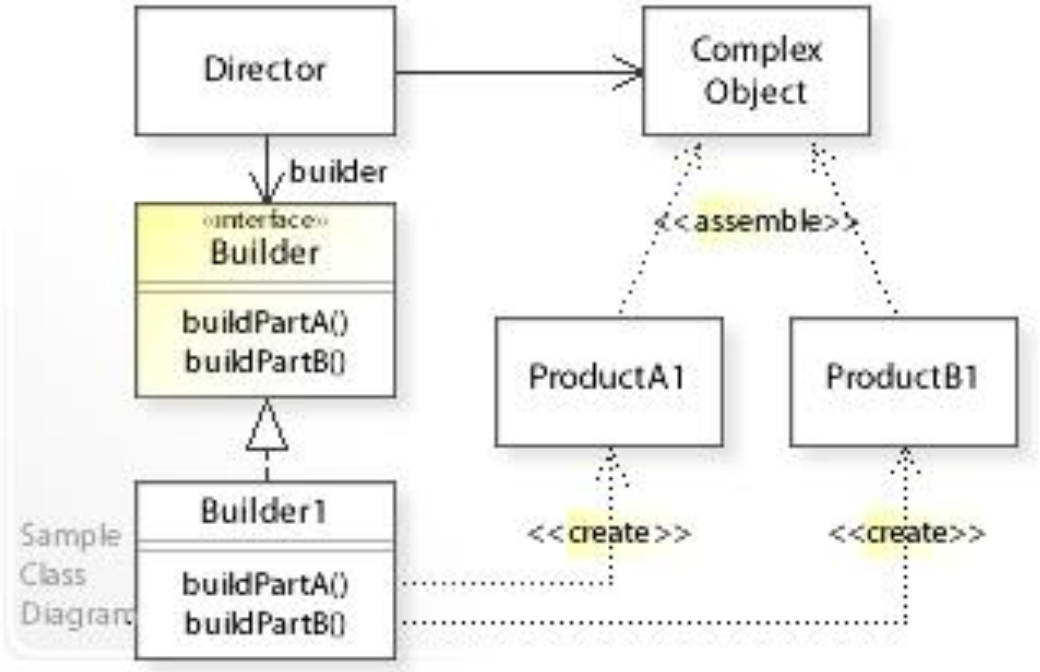
Mocheth Trianes 8-920-2031

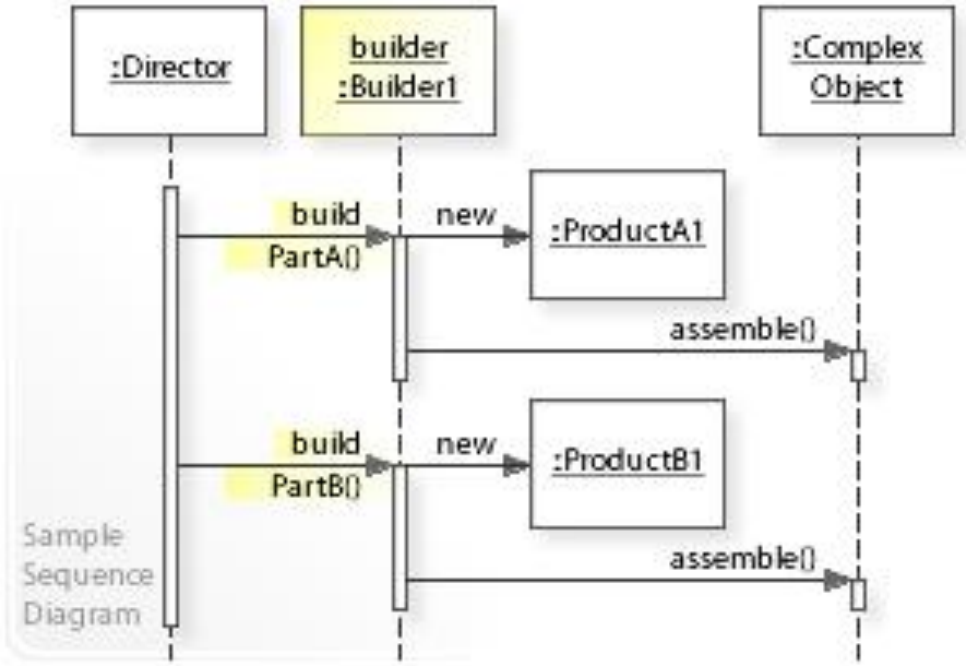




Builder

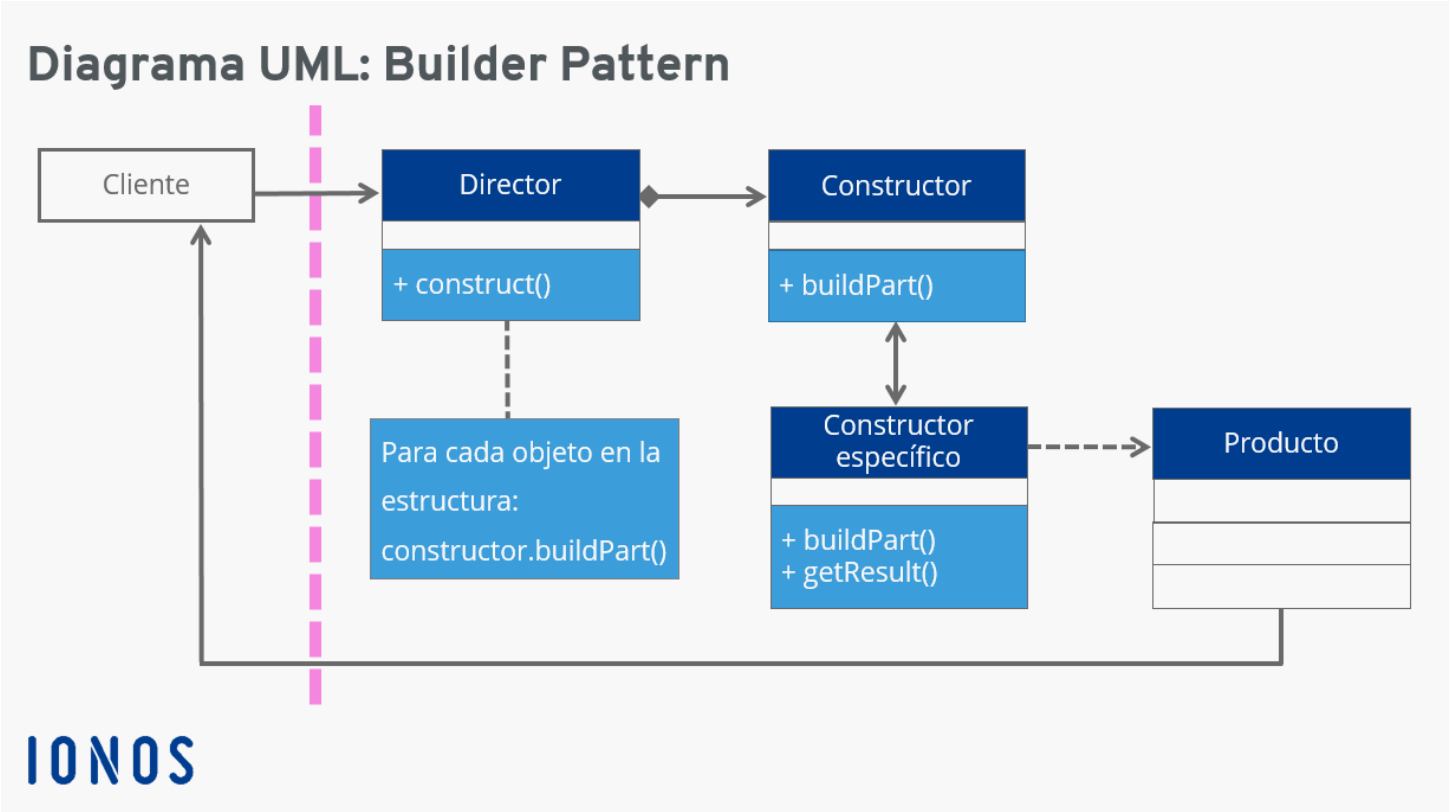
- Es un patrón de diseño creacional que nos permite construir objetos complejos paso a paso. El patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.





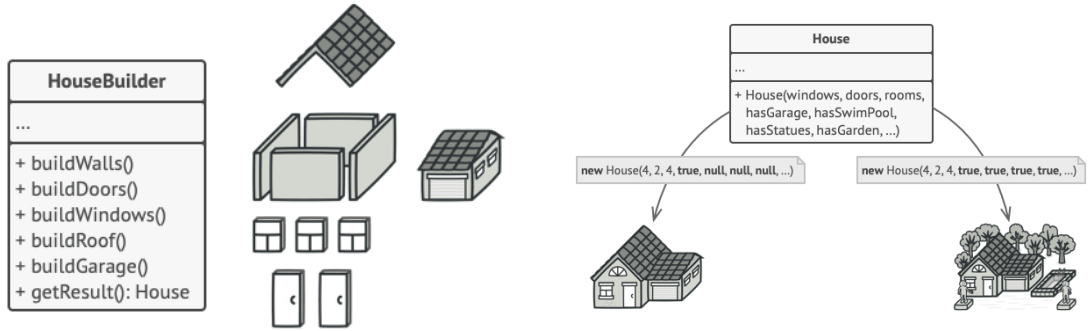
Builder: Funcionalidad

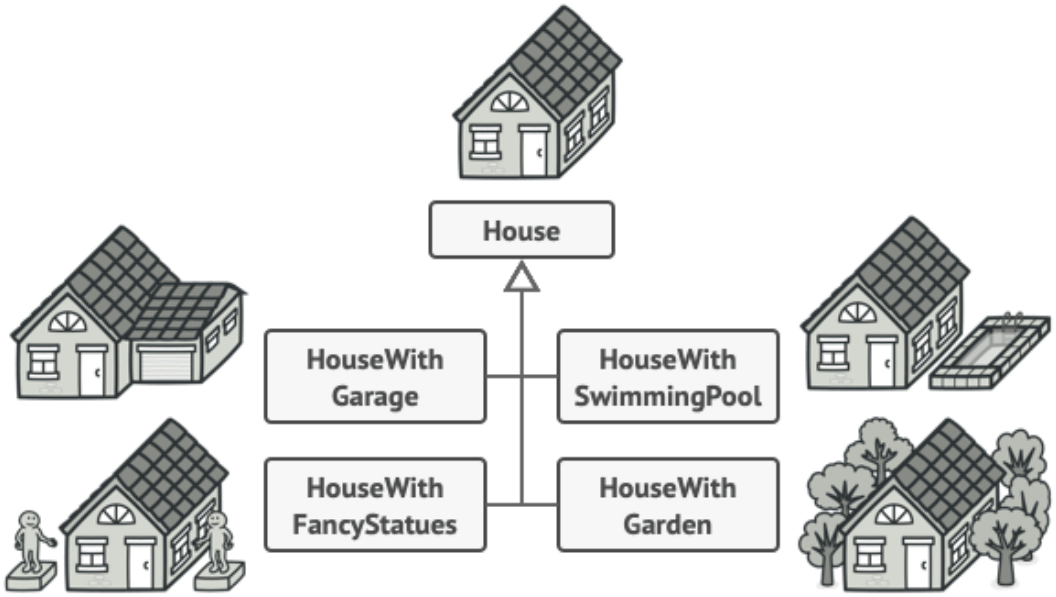
- Director: construye el objeto complejo con la interfaz del constructor.
- Builder: ofrece una interfaz para crear los componentes de un objeto (o producto) complejo.
- Specific Builder: crea las partes del objeto complejo, define (y gestiona) la representación del objeto y mantiene la interfaz de salida del objeto.
- Producto: es el resultado de la "actividad" del Builder Pattern, es decir, el objeto que se construye.



Builder: Ejemplo

- Se debe crear un objeto Casa. Para construir una casa sencilla, debemos construir cuatro paredes y un piso, así como instalar una puerta, colocar un par de ventanas y ponerle un tejado. Pero ¿qué pasa si quieres una casa más grande y luminosa, con un jardín y otros extras (como sistema de calefacción, instalación de fontanería y cableado eléctrico)





Builder: Ventajas y Desventajas

Ventajas

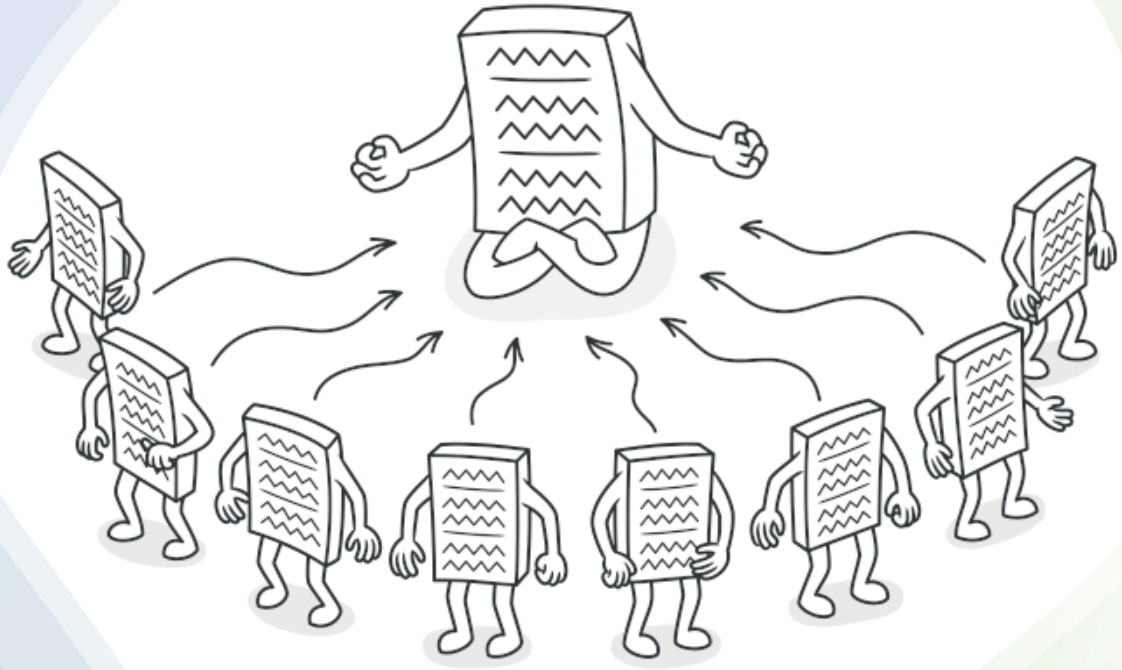
- La construcción y la representación (salida) se incorporan por separado.
- Las representaciones internas del constructor están ocultas para el director.
- El proceso de construcción lo controla explícitamente el director.

Desventajas

- Dificultad en hacer cambios en procesos básicos.

Singleton

- El propósito de este patrón es evitar que sea creado más de un objeto por clase. Esto se logra creando el objeto deseado en una clase y recuperándolo como una instancia estática.

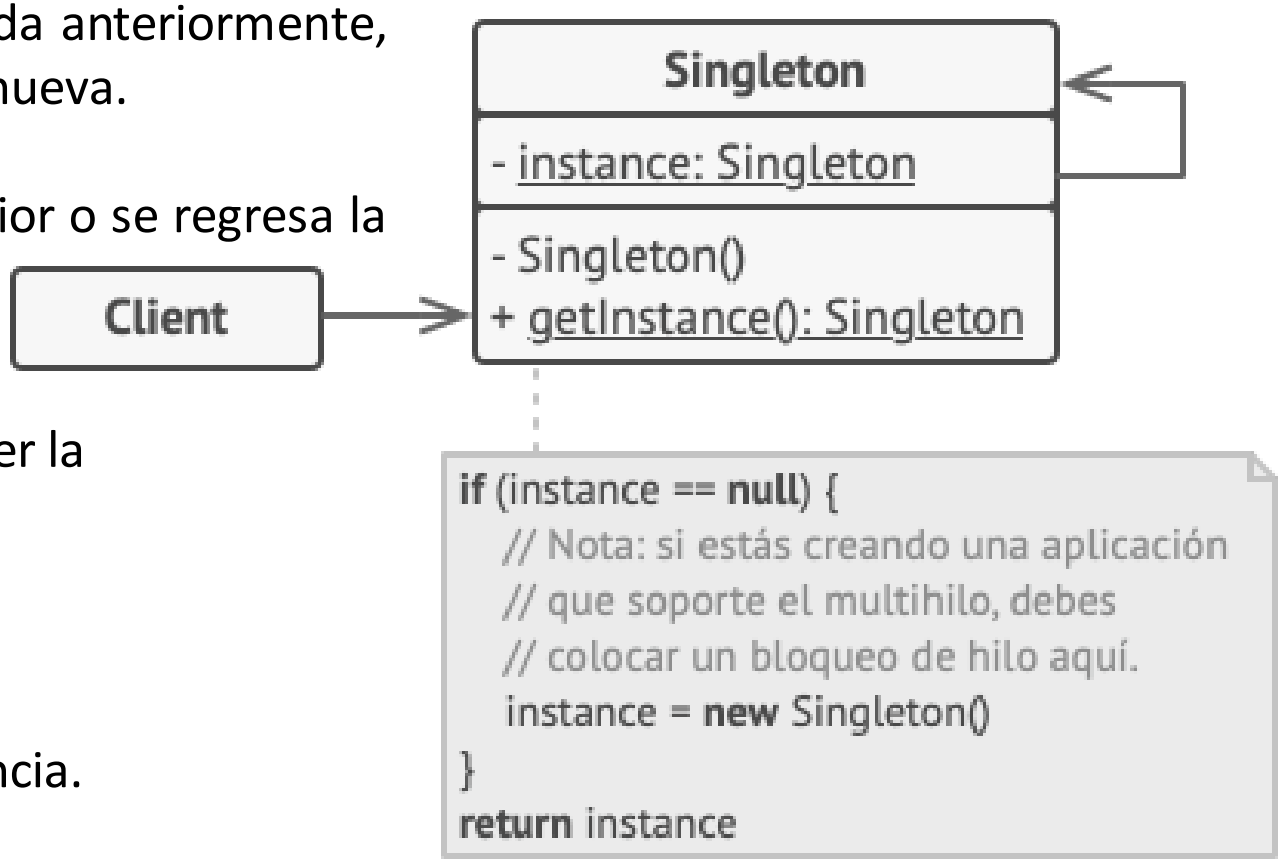


Funcionamiento:

1. El cliente solicita la instancia al Singleton mediante el método estático getInstance.

2. El Singleton validará si la instancia ya fue creada anteriormente, de no haber sido creada entonces se crea una nueva.

3. Se regresa la instancia creada en el paso anterior o se regresa la instancia existente en otro caso.



Ventajas:

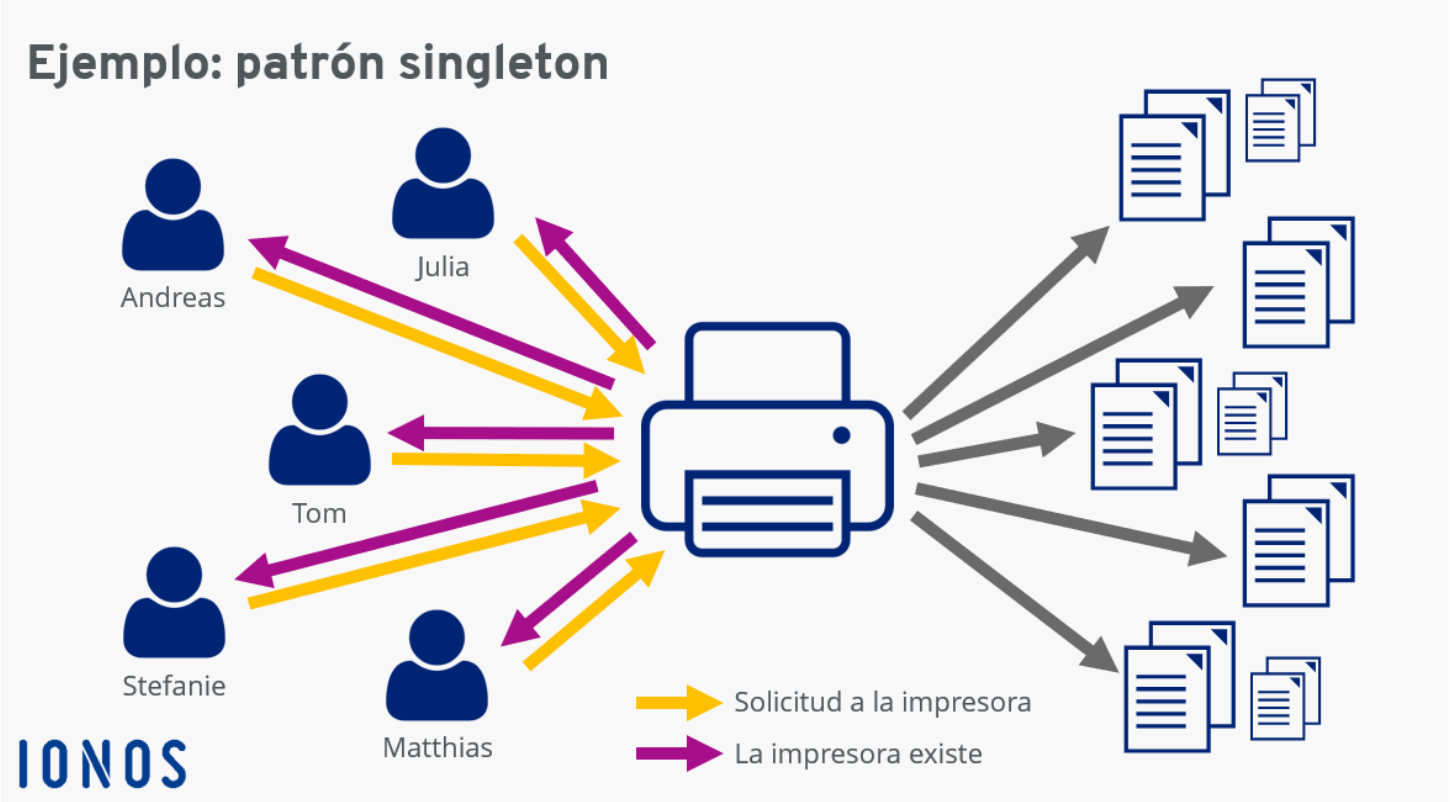
- La clase es la que se encarga de manejar y proveer la instancia.

- Es uno de los patrones de diseño más sencillos.

- Puede configurarse para tener más de una instancia.

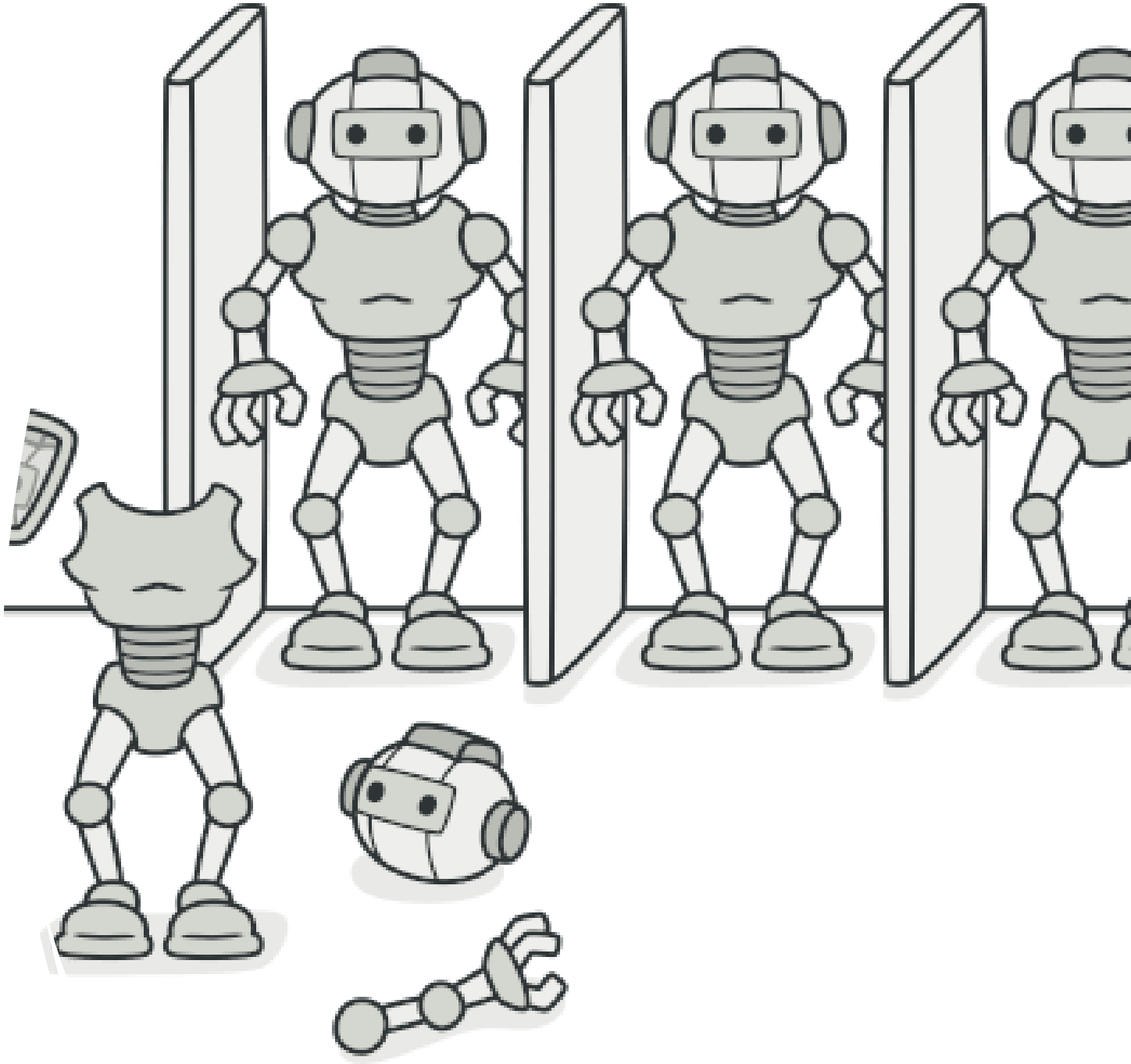
Ejemplo: Singleton

Singleton se utiliza en el software de gestión de la impresora para procesar las consultas sin modificarlas y, en última instancia, imprimirlas.



Prototype

- El patrón Prototype se basa en la idea de crear nuevos objetos mediante la clonación de un objeto existente, en lugar de crearlos desde cero.

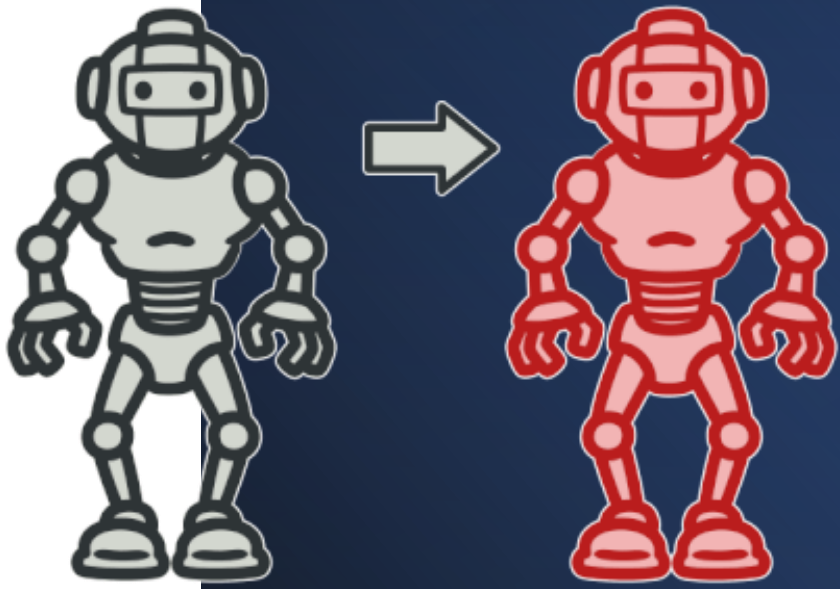


Prototype

- Este patrón es especialmente útil cuando tenemos objetos complejos que pueden ser costosos de crear. En lugar de crear un nuevo objeto desde cero, simplemente clonamos el objeto existente y hacemos las modificaciones necesarias en la copia.

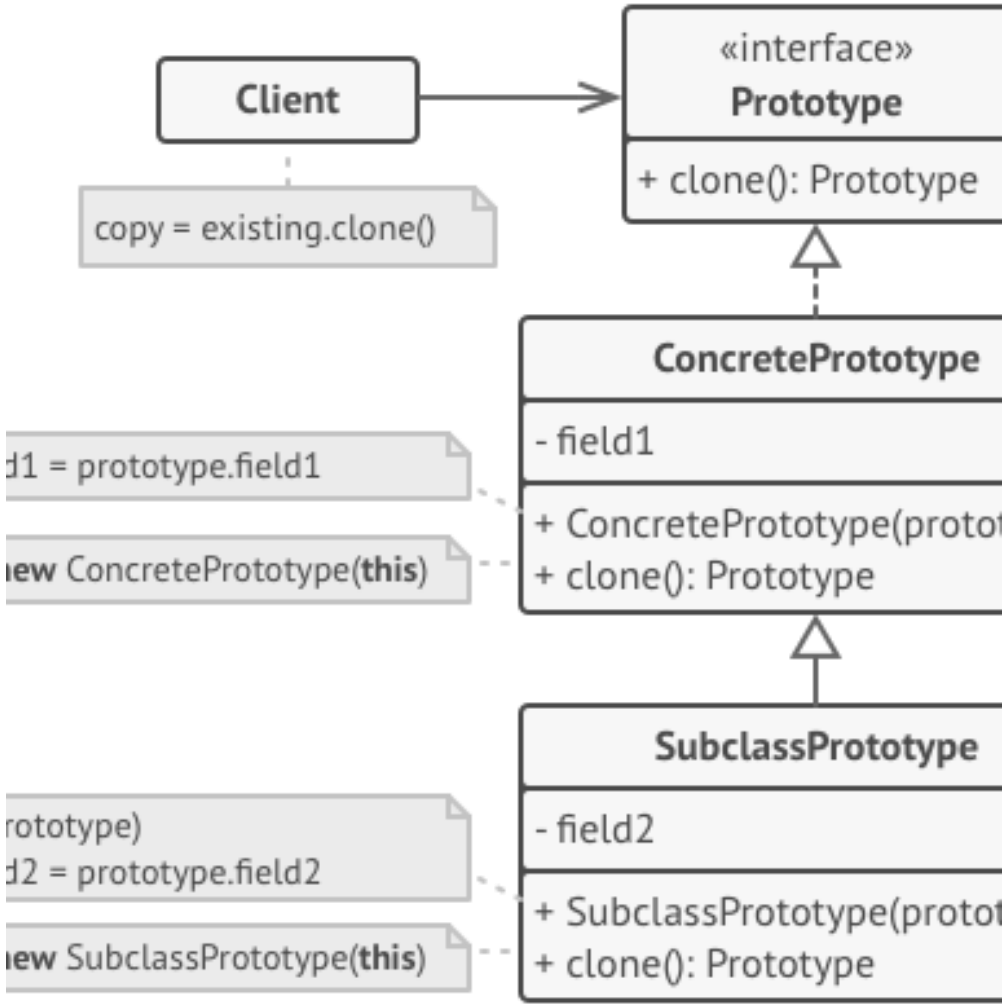
Ejemplo: Prototype

- El patrón Prototype también es útil en situaciones en las que necesitamos crear objetos que sean variaciones de un prototipo existente. Por ejemplo, si estamos construyendo un videojuego y queremos generar enemigos con diferentes habilidades y atributos, podemos tener un prototipo base de enemigo y clonarlo para crear distintas variantes con habilidades únicas.



Componentes que conforman el patrón

- **El Cliente:** puede producir una copia de cualquier objeto que siga la interfaz del prototipo.
- **La interfaz Prototipo**: declara los métodos de clonación.
- **La clase Prototipo Concreto:** implementa el método de clonación. Además de copiar la información del objeto original al clon, este método también puede gestionar algunos casos extremos del proceso de clonación, como, por ejemplo, clonar objetos vinculados, deshacer dependencias recursivas, etc.



Ventajas

- **Creación eficiente de objetos:** Una de las principales ventajas del patrón Prototype es que permite la creación eficiente de objetos complejos.

Desventajas

- **Impacto en el rendimiento:** Dependiendo de la implementación, la clonación de objetos puede afectar el rendimiento del sistema, especialmente si los objetos son grandes y la clonación es intensiva.