Patrones de Diseño: Patrones de Creación. Tema 3-5: Prototype

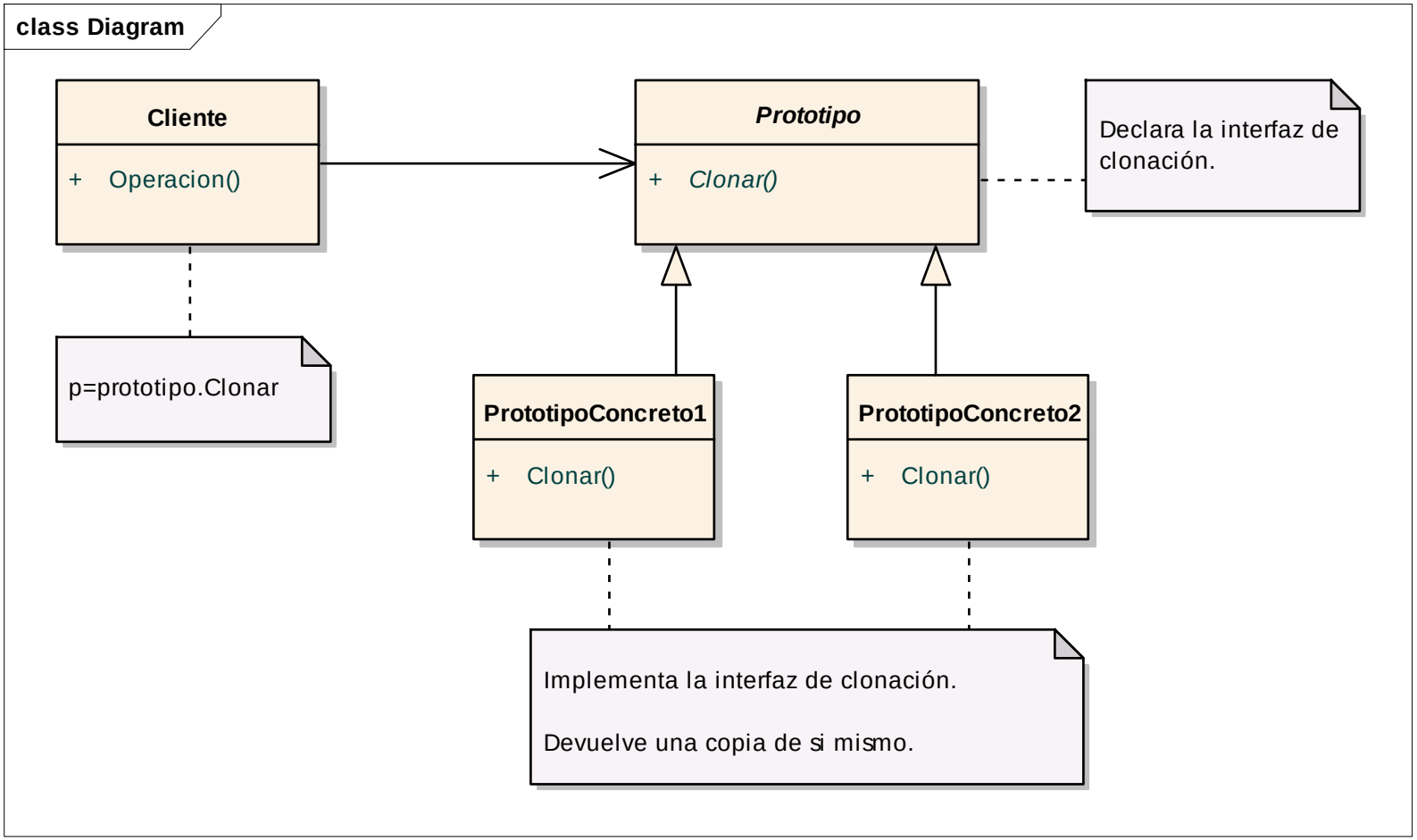
Descripción del patrón

- **Nombre:**
  - Prototipo
- **Propiedades:**
  - Tipo: creación
  - Nivel: objeto, clase única
- **Objetivo o Propósito:**
  - Especifica los tipos de objetos a crear por medio de una instancia prototípica, y crea nuevos objetos copiando dicho prototipo.
  - Este patrón se usa en los casos en los que crear una instancia de una clase sea un proceso muy complejo y requiera mucho tiempo. Lo que hace es crear una instancia original, y, cuando necesitemos otra, en lugar de volver a crearla, copiar esa original y modificarla.

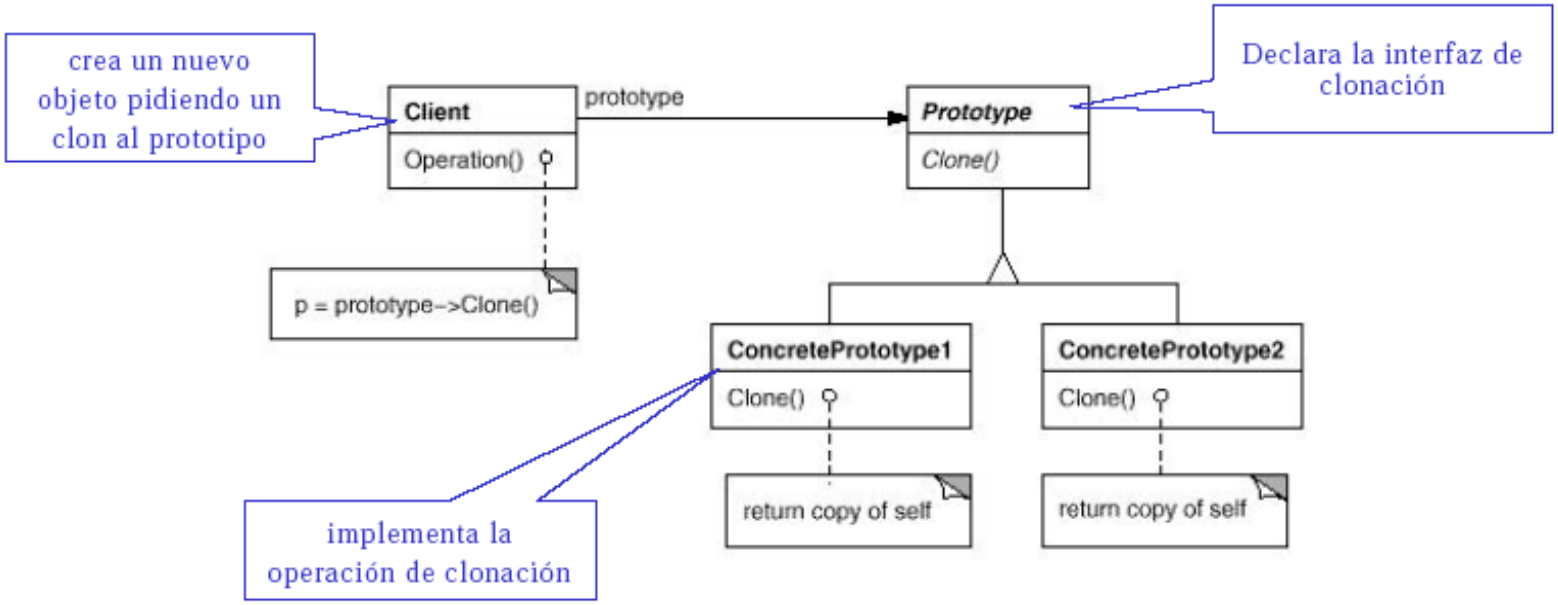
Aplicabilidad

- Use el patrón Prototype cuando:
  - Un sistema deba ser independiente de cómo se crean, se componen y se representan sus productos.
  - Cuando las clases a instanciar sean especificadas en tiempo de ejecución.
  - Para evitar construir una jerarquía de clases de fábrica paralela a la jerarquía de clases de los productos.
  - Cuando las instancias de una clase puedan tener uno de entre sólo unos pocos estados diferentes. Puede ser más adecuado tener un número equivalente de prototipos y clonarlos, en vez de crear manualmente instancias de la clase cada vez con el estado apropiado.
- Ejemplo: Un listado muy largo de una base de datos ordenado por distintos criterios ¿Qué es mejor?, ¿leer de nuevo la base de datos entera o hacer una copia de la primera lectura y ordenarla? Obviamente lo más adecuado es la segunda.

Estructura


class Diagram
Cliente
+ Operacion()
Prototipo
+ Clonar()
Declara la interfaz de clonación.
p=prototipo.Clonar
PrototipoConcreto1
+ Clonar()
PrototipoConcreto2
+ Clonar()
Implementa la interfaz de clonación.
Devuelve una copia de si mismo.

Estructura


crea un nuevo objeto pidiendo un clon al prototipo
Client
Operation()
prototype
Prototype
Clone()
Declara la interfaz de clonación
p = prototype->Clone()
ConcretePrototype1
Clone()
ConcretePrototype2
Clone()
return copy of self
return copy of self
implementa la operación de clonación

Estructura. Participantes

- **Prototipo**: Declara una interfaz para clonarse. Esta clase define la interfaz para crear clones de sí mismo. Típicamente esto involucra definir una función clone() que devuelve una copia del objeto original.

- **PrototipoConcreto**: Un Prototipo Concreto es una clase que implementa la operación de clonar definida en la clase Prototipo. Esto involucra copiar los valores y el estado del objeto original.

- **Cliente**: Crea un nuevo objeto pidiéndole a un prototipo que se clone.

Estructura. Variaciones

- **Variaciones del patrón:**

- **Constructor de copia:** Este tipo de constructores toma una instancia de la misma clase como parámetro y devuelve una nueva copia con los mismos valores que el parámetro. Es posible tener un constructor al que se le pasan dos parámetros, el objeto a ser copiado y un valor booleano para indicar si debe aplicarse copia superficial o profunda.
- **Método Clone:** Java define un método en la clase Object llamado clone para hacer copias de objetos.

Método clone

- Todas las clases en Java heredan un método de la clase *Object* llamado *clone*. Un método *clone* de un objeto retorna una copia de ese objeto. Esto solamente se hace para instancias de clases que dan permiso para ser clonadas.

- El método clone() es declarado “protected” (por defecto sólo puede ser llamado por el objeto que lo contenga, un objeto en el mismo paquete, u otro objeto del mismo tipo o subtipo). Si necesitamos hacer que un objeto pueda ser clonado por cualquiera, tenemos que redefinir su método clone() y hacerlo público.

- Una clase da permiso para que su instancia sea clonada si, y solo si, ella implementa el interface *Cloneable*.

Método clone

- Hay dos estrategias básicas para implementar la operación *clone*:

1. **Copia superficial** significa que las variables de los objetos clonados contienen los mismos valores que las variables del objeto original y que todas las referencias al objeto son a los mismos objetos. Es decir, la copia superficial copia solamente el objeto que será clonado, no los objetos a los que se refiere.
2. **Copia profunda** significa que las variables de los objetos clonados contienen los mismos valores que las variables del objeto original, excepto que estas variables que se refieren a objetos realizan copias de los objetos referenciados por el objeto original. Es decir, la copia profunda copia el objeto que será clonado y los objetos a los que se referencia.

- Implementar la copia profunda puede ser delicado. Se necesita decidir si se quiere hacer copia profunda o superficial de los objetos copiados indirectamente. También hay que tener cuidado con el manejo de las referencias circulares.

Consecuencias

- Como los otros patrones de creación de objetos (Abstract Factory y Builder), el Prototipo **aísla las clases de productos concretos del cliente**. Esto reduce el número de clases conocidas por el cliente y le permite trabajar con diferentes clases concretas sin ser modificado.
- Haciendo uso del patrón Prototipo, **podemos añadir y quitar productos en tiempo de ejecución**, según se vayan necesitando, mediante la clonación. Podemos ajustar la representación interna de los valores de una clase en tiempo de ejecución basándonos en las condiciones del programa. Además, podemos especificar nuevos objetos en tiempo de ejecución, sin crear una proliferación de clases y estructuras de herencia.

Consecuencias

- Otra manera de crear nuevos objetos es **variando sus estructuras**. Si implementamos clone() como copia profunda, se pueden añadir diferentes estructuras al conjunto de prototipos disponibles.
- Si tenemos clases de prototipo que copiar implica que tenemos el acceso suficiente a sus valores o métodos para cambiarlas después de ser clonadas. Esto puede requerir añadir métodos de acceso a los valores de las clases de estos prototipos para que podamos modificar los valores una vez que hayamos clonado la clase.
- Podemos crear un **registro de clases prototípicas** que pueden ser clonadas y pedir al objeto registro una lista de los posibles prototipos. A lo mejor podemos clonar una clase existente mejor que escribirla desde cero.

Consecuencias

- **Reduce la herencia.** El patrón Factory Method suele producir una jerarquía de clases Creador que es paralela a la jerarquía de clases de productos. El patrón Prototype permite clonar un prototipo en vez de decirle a un método de fabricación que cree un nuevo objeto. Por tanto no es necesaria una jerarquía de clases creador.

- El principal **inconveniente** del patrón Prototype es que cada subclase de Prototipo debe implementar la operación clonar, lo cual puede ser difícil. Por ejemplo, si las clases ya existen, es difícil añadir el método clonar, sobre todo si incluyen objetos que no pueden clonarse o que tienen referencias cíclicas.

Patrones relacionados

- **Abstract Factory**. Una factoría abstracta podría almacenar un conjunto de prototipos a partir de los cuales clonar y devolver objetos de productos.
- **Factory Method.** El patrón Factory Method podría ser una alternativa al patrón Prototipo cuando la paleta de objetos prototípicos no contenga más de un objeto.
- **Builder.** El Builder se centra en la creación paso a paso de objetos complejos. Se crea una clase Builder por cada clase de producto que encapsula la complejidad de crear el objeto. La ventaja del Prototipo es, a parte de la necesidad de sólo una clase “factory”, el hecho de que la lógica de la creación está unida directamente a cada clase de producto.
- **Singleton.** Puede ser usado en la implementación del patrón Prototipo.

Conclusiones

- Como otros patrones de creación, el patrón Prototipo como mejor se aplica es en los casos en los que el sistema debe ser independiente de cómo sus objetos son creados, compuestos y representados.
- El patrón Prototipo es útil en los casos en los que la construcción e inicialización de un objeto resulta más compleja o consume más tiempo que el que supone clonar un objeto existente.
- Permite añadir y quitar clases de productos en tiempo de ejecución clonándolos según se vayan necesitando. Además podemos especificar nuevos objetos en tiempo de ejecución sin crear una proliferación de clases y de estructuras de herencia.
- La utilidad de este patrón, puede aumentar si lo combinamos con el patrón Abstract Factory, o si creamos un registro en el que almacenar y gestionar todos los prototipos de un sistema.

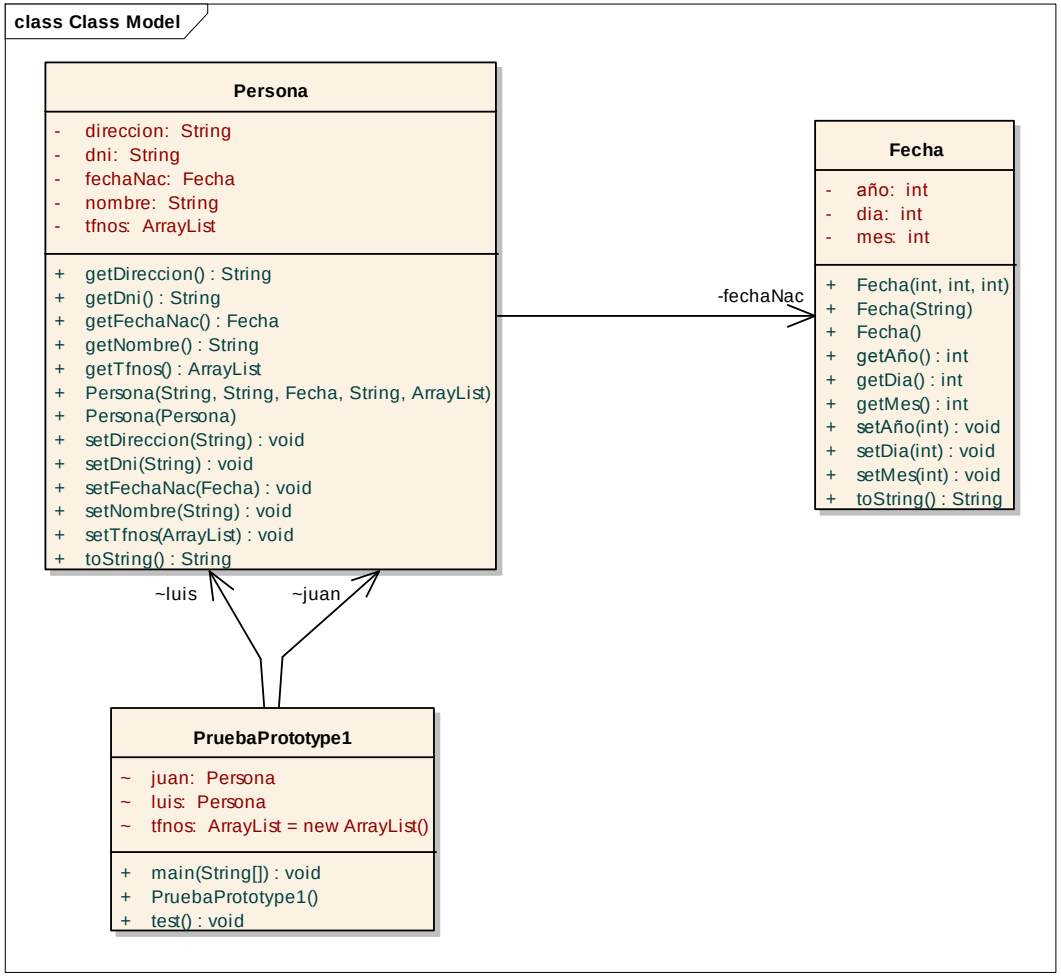
Código de ejemplo

CLONACIÓN PERSONAS

Código de ejemplo. V1

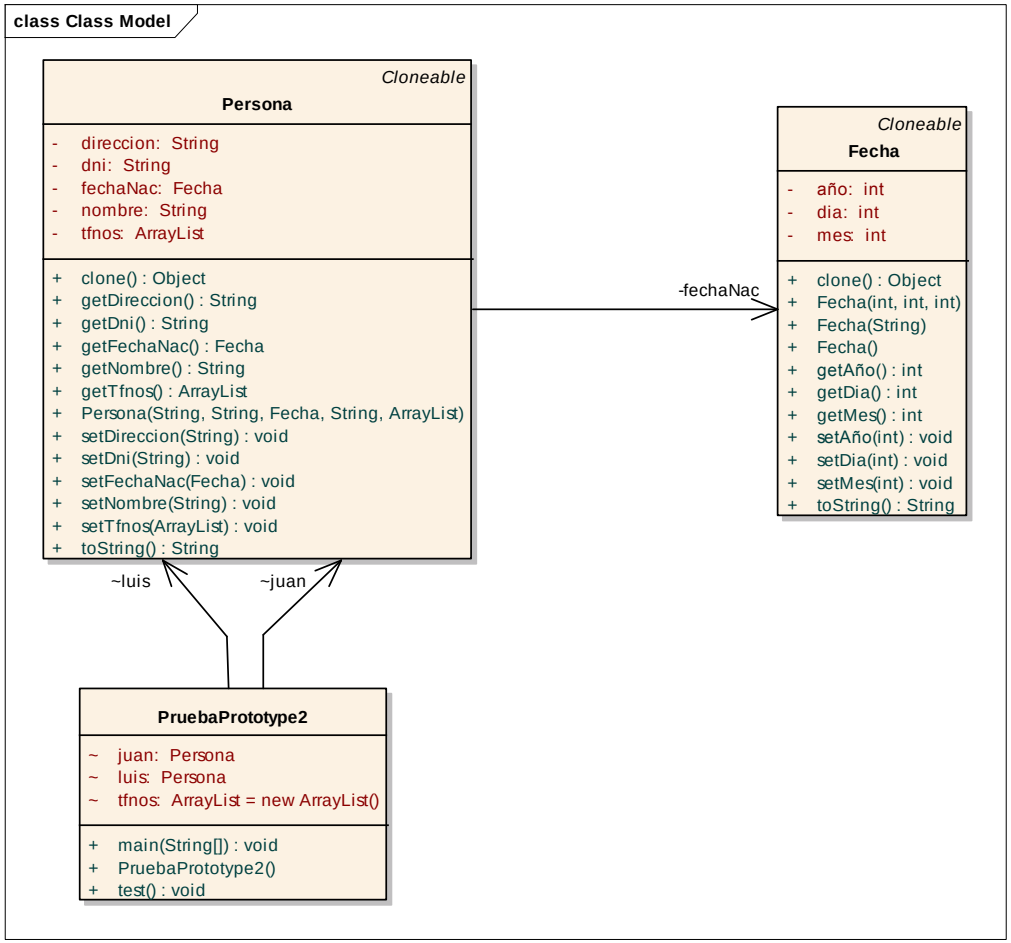
CONSTRUCTOR

DE COPIA:



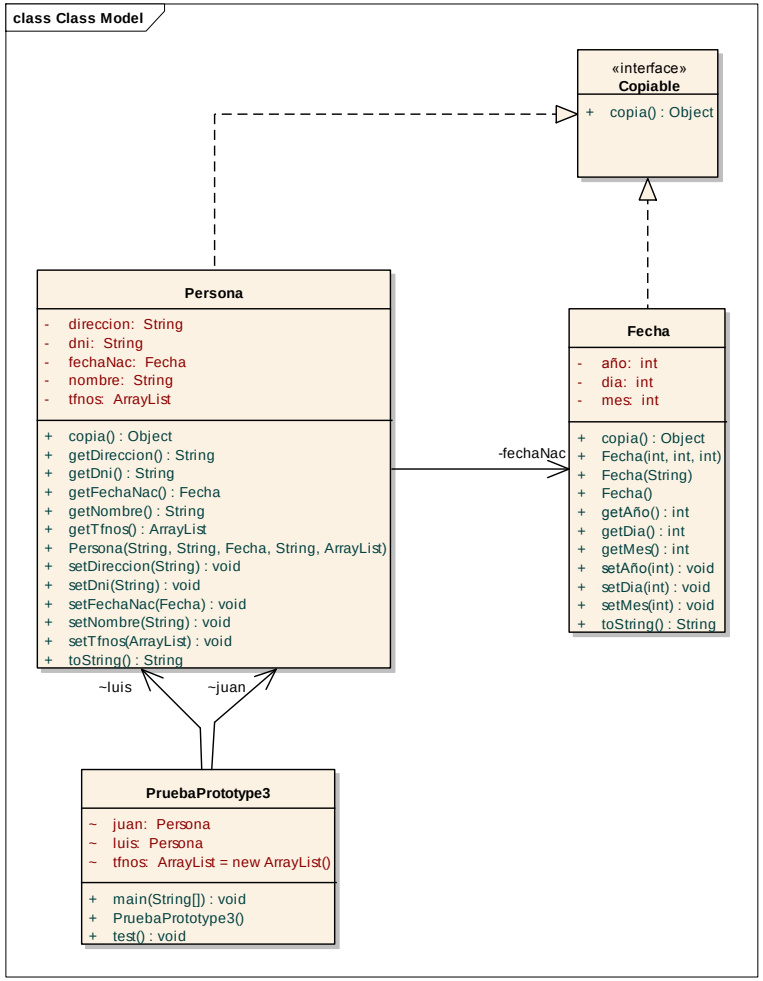
Código de ejemplo. V2

MÉTODO CLONE:



Código de ejemplo. V3

MÉTODO COPIA:



Código de ejemplo. V3

- Identificamos a continuación los elementos del patrón:
  - Prototipo: interfaz Copiable.
  - PrototipoConcreto: Persona, Fecha.
  - Cliente: PruebaPrototipo3.