

Hillary Malespín Ulloa
Jurgenn Morales Jiménez

Jurgenn Morales Jiménez





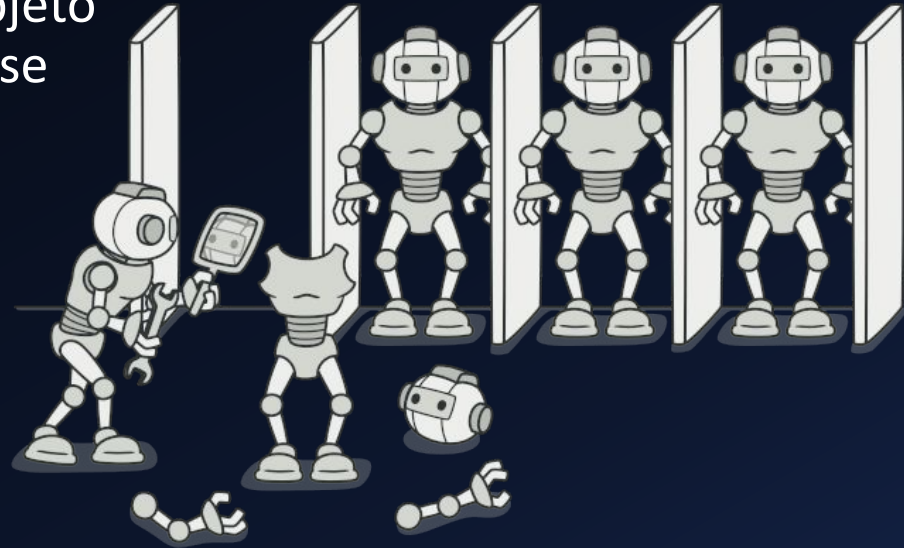
¿En qué consiste?

Este patrón usa una instancia de prototipo para definir un tipo de objeto y luego crear un nuevo objeto. Los objetos copian este prototipo.

El patrón Prototype delega el proceso de clonación al propio objeto clonado. El modelo declara una interfaz común para todos los objetos que admiten la clonación.

Esta interfaz le permite clonar un objeto sin que su código se conecte a la clase de este objeto.

Estas interfaces suelen contener un único método de clonación.





¿Cuándo utilizarlo?

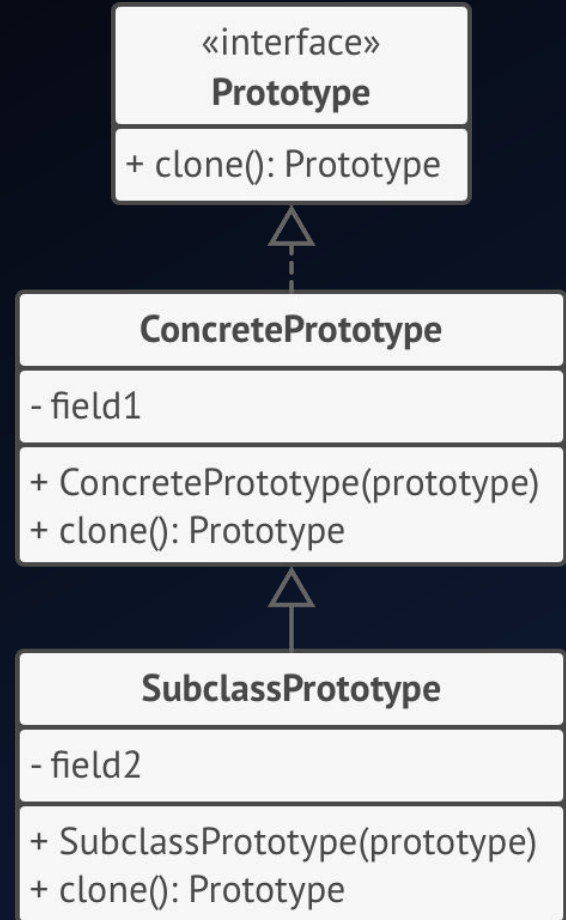
Se debería utilizar el patrón cuando un sistema debe ser independiente de cómo se crean, componen y representan sus productos, y:

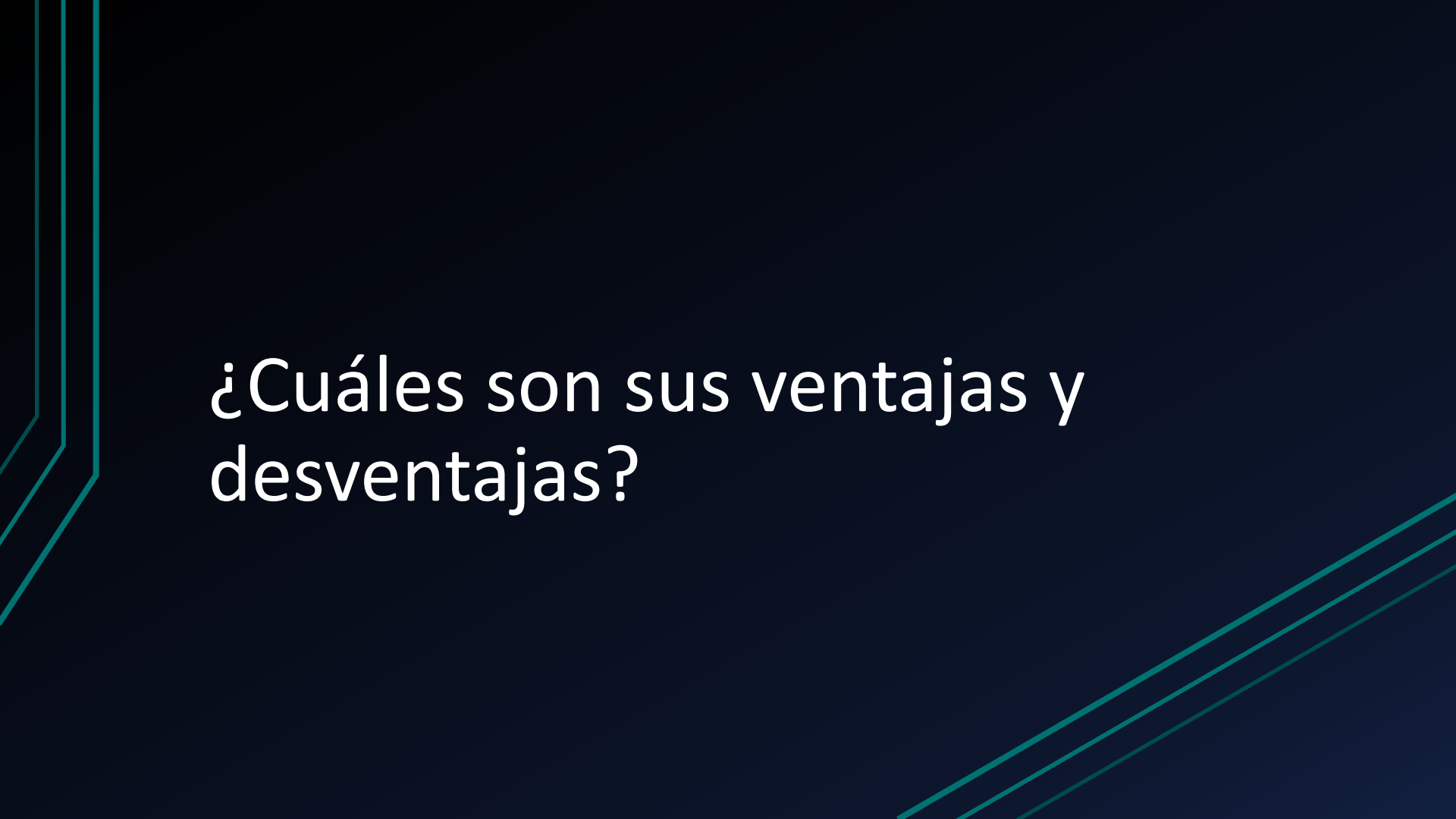
- Cuando las clases a instanciar se especifican en tiempo de ejecución, por ejemplo, mediante carga dinámica.
- Para evitar construir una jerarquía de clases de fábricas que sea paralela a la jerarquía de clases de productos.
- Cuando las instancias de una clase pueden tener una de sólo unas pocas combinaciones diferentes de estado.
 - Puede ser más conveniente instalar un número correspondiente de prototipos y clonarlos en lugar de instanciar la clase manualmente, cada vez con el estado apropiado.



¿Cómo se implementa?

1. Crear una interfaz de prototipo y declarar el método clonar o, en caso de utilizar herencia, agregar el método a la clase padre.
2. Definir en las clases prototipo un constructor alternativo que reciba un objeto de su misma clase como parámetro.
 - a. El constructor debe copiar los valores de todos los atributos del objeto que recibe a la nueva instancia.
 - b. Si se clona una subclase se debe invocar al constructor de la clase padre para que copie los atributos privados de esta.
3. Sobreescibir el método clonar en todas las clases.
4. Opcionalmente se puede crear un registro o catálogo para almacenar prototipos utilizados frecuentemente.





¿Cuáles son sus ventajas y desventajas?

Ventajas

- Permite clonar objetos sin acoplarlos a clases concretas.
- Evita la repetición de código para inicializar objetos.
- Permite crear objetos complejos de manera más sencilla.
- Permite definir y eliminar configuraciones de objetos durante la ejecución.

Desventajas

- Clonar objetos con atributos complejos puede ser difícil de implementar.
 - Por ejemplo atributos que no se puedan clonar o tengan una referencia circular.

Ejemplo en Java

Creación de zombis para un videojuego

Interfaz Prototipo

```
package poo.demoprototype;

public interface Prototipo {

    public Prototipo clonar();

}
```

Clase Zombi

```
package poo.demoprototype;

public class Zombi implements Prototipo {

    private int salud, ataque;

    public Zombi(int salud, int ataque) {
        this.salud = salud;
        this.ataque = ataque;
    }

    public Zombi(Zombi prototipo) {
        this.salud = prototipo.salud;
        this.ataque = prototipo.ataque;
    }

    @Override
    public Prototipo clonar() {
        return new Zombi(this);
    }

    public void mostrarDatos() {
        System.out.println("Zombi: Salud:" + salud + ", Ataque:" + ataque);
    }

    public void setSalud(int salud) {
        this.salud = salud;
    }

}
```

Clase RegistroZombis

```
package poo.demoprototype;

import java.util.HashMap;

public class RegistroZombis {

    private HashMap<String, Zombi> listaPrototipos;

    public RegistroZombis() {
        listaPrototipos = new HashMap<>();
    }

    public void agregarPrototipo(String llave, Zombi valor) {
        listaPrototipos.put(llave, valor);
    }

    public Zombi obtenerZombi(String llave) {
        return (Zombi) listaPrototipos.get(llave).clonar();
    }
}
```

Clase Demo

```
package poo.demoprototype;

public class Demo {

    public static void main(String[] args) {
        // Crea un registro de prototipos de zombi
        RegistroZombis registro = new RegistroZombis();

        // Crea un par de zombis con valores especificos
        Zombi pequeno = new Zombi(25, 5);
        Zombi normal = new Zombi(50, 10);

        // Guarda los zombis creados en el registro de prototipos.
        // Los guarda con una llave representativa de sus valores.
        registro.agregarPrototipo("pequeño", pequeno);
        registro.agregarPrototipo("normal", normal);

        // Crea unos clones a partir de los prototipos y muestra sus datos
        Zombi clon1 = registro.obtenerZombi("normal");
        Zombi clon2 = registro.obtenerZombi("pequeño");
        clon2.setSalud(15); // Reduce la salud de un zombi
        Zombi clon3 = registro.obtenerZombi("pequeño");

        clon1.mostrarDatos();
        clon2.mostrarDatos();
        clon3.mostrarDatos();
    }
}
```

Resultado de ejecución

Zombi: Salud:50, Ataque:10

Zombi: Salud:15, Ataque:5

Zombi: Salud:25, Ataque:5

Referencias

- Refactoring.Guru. (s.f.). *Prototype*.
<https://refactoring.guru/es/design-patterns/prototype>
- Gamma, E., Helm, R., Johnson, R. y Vlissides, J. (31 de octubre de 1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education.