PROTOTYPE

JOSHUA ABRIL

PATRONES DE SOFTWARE

UNINPAHU

2025

1) ¿Qué es el patrón de diseño Prototype?:

El patrón **Prototype** permite crear nuevos objetos copiando una instancia existente **(prototipo)** en lugar de construirlos desde cero, lo cual puede ser costoso o complejo.

2) ¿Cuál es el objetivo del patrón Prototype?:

Facilitar la **creación eficiente de objetos** mediante clonación, especialmente cuando el proceso de instanciación es caro, lento o involucra mucha configuración.

3) ¿Qué problema resuelve el patrón Prototype?:

Evita la creación repetitiva desde cero y permite **duplicar objetos personalizados o configurados**, ahorrando tiempo y recursos.

4) ¿En qué situaciones es preferible usar Prototype en lugar de new?:

- Cuando crear un objeto con **new** es costoso.
- Cuando se necesita **copiar objetos configurados dinámicamente**.
- Cuando se desea mantener la **flexibilidad al clonar objetos sin conocer su tipo exacto**.

5) ¿Qué se necesita para que un objeto sea clonable?:

 Implementar una interfaz o método de clonación, como **Cloneable** en **Java** o **ICloneable** en **C#.**
 Definir el método **clone()** o **Clone()** correctamente para evitar errores y controlar el tipo de copia (**superficial o profunda**).

6) ¿Cuál es la diferencia entre una copia superficial y una copia profunda en Prototype?:

- **Profunda:** Copia **todo el objeto y sus subobjetos**, creando duplicados independientes.
- **Superficial:** Copia los **atributos de primer nivel**, referencias incluidas.

7) Una ventaja del patrón Prototype:

Permite **crear múltiples instancias rápidamente**, especialmente cuando el tipo exacto del objeto debe mantenerse y su configuración es compleja o dinámica.

8) ¿Cómo se implementa el patrón Prototype en Java o C#?:

- **Java:**

```
class Documento implements Cloneable {
    public String texto;

    public Documento(String texto) {
        this.texto = texto;
    }

    public Documento clone() throws CloneNotSupportedException {
        return (Documento) super.clone();
    }
}
```

- **C#:**

```
class Documento : ICloneable {
    public string Texto;

    public object Clone() {
        return this.MemberwiseClone();
    }
}
```

9) ¿Qué riesgos pueden surgir al usar el patrón Prototype?:

- **Clonación incorrecta** (copias superficiales que comparten referencias no deseadas).
- **Objetos complejos** que no implementan bien la clonación.
- Posibles **errores al clonar subobjetos** o mantener la integridad de datos.

10) Un ejemplo práctico del uso del patrón Prototype:

- **Ejemplo:** En un **software** de diseño gráfico, el **usuario** crea una figura personalizada con colores, bordes y sombras. Para duplicarla sin repetir todo el proceso, se usa **clone()**.

- **Código:**

```
class Figura implements Cloneable {

    private String tipo;

    private String color;

    private String sombra;
```

```
    public Figura(String tipo, String color, String sombra) {
        this.tipo = tipo;
        this.color = color;
        this.sombra = sombra;
    }

    public void mostrar() {
        System.out.println("Figura: " + tipo + ", Color: " + color + ", Sombra: " + sombra);
    }

    @Override
    public Figura clone() {
        try {
            return (Figura) super.clone();
        } catch (CloneNotSupportedException e) {
            System.out.println("Error al clonar la figura.");
            return null;
        }
    }
```

```
}


public class EjemploPrototype {
    public static void main(String[] args) {
        Figura original = new Figura("Círculo", "Rojo", "Sombra suave");
        System.out.println("Original:");
        original.mostrar();


        Figura copia = original.clone();
        System.out.println("Copia:");
        copia.mostrar();
    }
}
```

- **Algoritmo:**

[Cliente]

|

v

[Figura (prototipo configurado)]

|

v

[Llama a clone()]

|

v

[Figura clonada (idéntica a la original)]

|

v

[Cliente usa o modifica la figura clonada]