2017

Diesño Orientado a Objetos

1595054
CARLOS ABEL LUGO GONZALEZ

FCFM | Ciudad Universitaria

¿Qué son y para qué sirven los patrones de diseño?

En desarrollo web tenemos dos opciones principales: Desarrollo Frontend y Desarrollo Backend. Algunas personas suelen preguntarse ¿es mejor el desarrollo frontend? ¿Es mejor el desarrollo backend?.

Los frontends tienden a ser programadores, pero hay diseñadores que también hacen frontend. Son los encargados de maquetar la estructura semántica del contenido (HTML), codificar el diseño en hojas de estilo (CSS) y agregar la interacción con el usuario (Javascript).

Un backend compone el acceso a bases de datos y generación de plantillas del lado del servidor. En backend se encargan de implementar MySQL, Postgres, SQL Server o MongoDB. Luego, un lenguaje como PHP o JSP, o frameworks como RoR, Django, Node.JS o .NET se conectan a la base de datos. Si te llama la atención el desarrollo backend, sigue leyendo este post.

CREACIÓN	ESTRUCTURALES	COMPORTAMIENTO
Abstract Factory (O)	Adapter (C)	Chain of responsibility (O)
Builder (O)	Bridge (O)	Command (O)
Factory Method (C)	Composite (O)	Interpreter (C)
Prototype (O)	Decorator (O)	Iterator (O)
Singleton (O)	Facade (O)	Mediator (O)
	Flyweight (O)	Memento (O)
	Proxy (O)	Observer (O)
		State (O)
		Strategy (O)
		Template Method (C)
		Visitor (O)

Ámbito: (C) Clase, (O) Objeto.

Cuando empezamos a desarrollar software, es común que cada quien utilice su propia lógica, conocimientos y experiencia para crear código. Y esto muchas veces resulta en desarrollos complejos que sólo su creador entiende. Pero, ¿es posible desarrollar un módulo que otro programador pueda aprovechar entender y mejorar? La respuesta está en los patrones de diseño. Estos básicamente son modelos muestra que sirven como guía para que los programadores trabajen sobre ellos

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

En resumen y de forma sencilla, un patrón de diseño es una manera de resolver un problema. Un patrón de diseño debe cumplir al menos con los siguientes objetivos.

1. Estandarizar el lenguaje entre programadores

2. Evitar perder tiempo en soluciones a problemas ya resueltos o conocidos
3. Crear código reusable (excelente ventaja)
4. Tipos de patrones de diseño

Los patrones de diseño se clasifican en tres tipos diferentes dependiendo del tipo de problema que resuelven. Estos pueden ser creacionales, estructurales y de comportamiento.

- Creacionales: Su objetivo es resolver los problemas de creación de instancia.

  - Singleton (Instancia única): Nos garantiza la existencia de una única instancia para una clase.

Un ejemplo de Singleton es:

```
public class SoyUnico {

    private String nombre;
    private static SoyUnico soyUnico;

    // El constructor es privado, no permite que se genere un constructor por defecto.
    private SoyUnico(String nombre) {
        this.nombre = nombre;
        System.out.println("Mi nombre es: " + this.nombre);
    }

    public static SoyUnico getSingletonInstance(String nombre) {
        if (soyUnico == null){
            soyUnico = new SoyUnico(nombre);
        }
        else{
            System.out.println("No se puede crear el objeto "+ nombre + " porque ya existe un objeto de la clase
SoyUnico");
        }

        return soyUnico;
    }

    // metodos getter y setter
}
```


- Prototype (prototipo): Clona las instancias ya existentes.

Un ejemplo a continuación:

```
var current = Object.prototype.valueOf;


// Como mi propiedad "-prop-value" es un atajo y no se encuentra siempre
// en la cadena de prototype, queremos modificar Object.prototype:
Object.prototype.valueOf = function() {
  if (this.hasOwnProperty('-prop-value')) {
    return this['-prop-value'];
  } else {
    // No parece que este objeto sea uno de los mios, por lo que recaeeremos
```

```
    // en el comportamiento por defecto lo mejor que podamos.
    // La llamada apply se comporta como el "super" en otros lenguages de programación.
    // A pesar de que valueOf() no tiene parametros, alguna otra llamada podria tenerlos.
    return current.apply(this, arguments);
  }
}
```

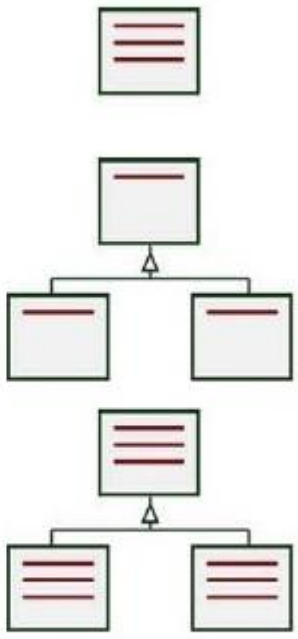
Estructurales: Su nombre es muy descriptivo, se ocupa de resolver problemas sobre la estructura de las clases. Por ejemplo:

Bridge (Puente) Separa la abstracción de la implementación.

Factory:

PATRON FACTORY

- Define una interface para crear objetos, dejando a las subclases decidir la clase específica. Permite delegar la responsabilidad de la instanciación a las subclases.



Simple Factory
Clase con la responsabilidad de crear objetos de otras clases. No delega en subclases y sus métodos pueden ser estáticos. Puede evolucionar a un Factory Method o Abstract Factory

Factory Method
Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue sus subclases la creación de objetos

Abstract Factory
Proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas

Ventajas: Factory Methods elimina la necesidad de asociar aplicación-clases específicas en el código. Sólo trabajamos con interfaces.

¿Cómo se documenta un patrón de diseño?

Pattern	**Nombre que se le da al patrón en este diseño concreto**
based on	Nombre de uno de los patrones de [GHJV03]
because	Fundamentación de la elección del patrón en términos de — Los cambios que este admite y los cambios probables anticipados en el diseño concreto — Las necesidades funcionales de alguna parte del sistema — Las restricciones de diseño que se deseen imponer
where	elemento_patrón **is** elemento_diseño elemento_patrón **is** elemento_diseño elemento_patrón **is** elemento_diseño ..... **is** .....
comments	Explicación coloquial de la relación entre los elementos del patrón y los elementos del diseño concreto; otros comentarios adicionales que ayuden a entender cómo se aplica el patrón de diseño.

Bibliografía

https://platzi.com/blog/patrones-de-diseno/

https://jarroba.com/patron-singleton-en-java-con-ejemplos/

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Object/prototype

https://danielggarcia.wordpress.com/2014/03/17/patrones-estructurales-iv-patron-bridge/