# fundamentos de POO. Polimorfismo.

Los patrones de diseño son el modelo de las soluciones a problemas comunes en el desarrollo de software. Estos modelos sirven como guía para que los programadores.

En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Debemos tener presente los siguientes elementos de un patrón: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios).

Los cuales se clasifican como se muestra a continuación:

* **Patrones Creacionales**: Inicialización y configuración de objetos.
* **Patrones Estructurales**: Separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan, para formar estructuras más grandes.
* **Patrones de Comportamiento**: Más que describir objetos o clases, describen la comunicación entre ellos.

## Marco Teórico.

### Introducción.

### Los patrones de diseño, design patterns, son métodos reusables para solucionar problemas de diseño comunes dentro de un contexto. Es una descripción o plantilla de cómo solucionar un problema que puede usarse en múltiples situaciones y lenguajes. Los patrones están formalizados como "mejores prácticas" que el desarrollador puede usar para resolver problemas comunes cuando diseña una aplicación. Esto no significa que tengan que aplicarse siempre, depende del criterio del desarrollador cuándo y cómo aplicarlos en una situación y proyecto concretos.

### Los patrones de diseño pueden mejorar el proceso y rendimiento del desarrollo. Un diseño de proyecto efectivo requiere considerar problemas que pueden no verse hasta después en la implementación. Reusar patrones de diseño ayuda a prevenir problemas que pueden causar mayores problemas y mejora la legibilidad del código para desarrolladores familiarizados con los patrones.

### A menudo la gente sólo entiende cómo aplicar determinados patrones para problemas concretos, esto es debido a que estas técnicas son difíciles de aplicar para un rango amplio de problemas. Los patrones de diseño proveen soluciones generales, documentadas en un formato que no requiere reglas específicas para problemas específicos. Los patrones más comunes van mejorando con el tiempo, haciéndolos más robutos y efectivos.

### Los patrones de diseño se dividen generalmente en:

### Creational patterns

Tratan la instancia de las clases. Estos patrones pueden dividirse después en class-creation patterns y object-creational patterns. Mientras que los class-creation patterns utilizan la herencia en el proceso de instanciación, object-creation patterns utilizan la delegación para terminar la tarea.

* **Abstract factory**. Crea una instancia de diferentes familias de clases.
* **Builder**. Separa la construcción de objetos de su representación.
* **Factory method**. Crea una instancia de varias clases derivadas.
* **Object pool**. Evita adquisiciones costosas y libera recursos reciclando objetos que no se usan.
* **Prototype**. Una instancia iniciada lista para ser copiada o clonada.
* **Singleton**. Una clase de la que sólo puede existir una instancia.

### Structural patterns

### Tratan la composición de clases y objetos. Los patrones structural class-creation utilizan la herencia para componer interfaces. Los patrones structural object-patterns definen formas de componer objetos para obtener nueva funcionalidad.

1. **Adapter**. Ajusta las interfaces de distintas clases para que coincidan.
2. **Bridge**. Separa la interface de un objeto de su implementación.
3. **Composite**. Una estructura en árbol de objetos simples y compuestos.
4. **Decorator**. Añade responsabilidades a objetos de forma dinámica.
5. **Facade**. Una clase simple que representa un subsistema entero.
6. **Flyweight**. Ligera instancia usada para que sea eficiente de compartir.
7. **Proxy**. Un objeto que representa a otro objeto.

### Behavioral patterns

Tratan la comunicación de las clases de los objetos, y por tanto tratan la comunicación entre objetos.

1. **Chain of responsibility.** Forma de pasar un request entre una cadena de objetos.
2. **Command.** Encapsula un command request como un objeto.
3. **Interpreter.** Forma de incluir elementos del lenguaje en un programa.
4. **Iterator.** Acceder a elementos de una colección de forma secuencial.
5. **Mediator.** Define una comunicación simplificada entre clases.
6. **Memento.** Captura y restaura el estado interno de un objeto.
7. **Null Object.** Diseñado para actuar como valor por defecto de un objeto.
8. **State.** Altera el comportamiento de un objeto.
9. **Strategy**. Encapsula un algoritmo dentro de una clase.
10. **Template method**. Aplaza los pasos exactos de un algoritmo a una subclase.
11. **Visitor**. Define una nueva operación a una clase sin cambios.

### Architectural patterns

Tienen una aplicación más amplia que las anteriores, pero tienen un rango más amplio de posibilidades. Son muy utilizados en los frameworks actuales de PHP.

1. **Front controller**. Proporciona una forma centralizada de manejar los requests.
2. **MVC. Model-View-Controller**. Divide una aplicación en tres partes interconectadas, separando las partes internas de la representación.
3. **ADR**. Action-Domain-Responder. Se plantea como un patrón MVC más refinado y orientado al desarrollo web.
4. **Service locator**. Emplea un registro central que devuelve información necesaria para tareas cuando se hacen peticiones.
5. **Active record**. Acceso de datos de una base de datos mediante un objeto.
6. **Publish-subscribe**. Forma de notificar los cambios en un número de clases.
7. **Inversion of control**. Diseño que invierte el funcionamiento tradicional en el que el código customizado llama a librerías reusables.

## conclusiones

Como verán, si es que este artículo logró ilustrar sobre las distintas categorías y tipos de patrones de diseño, no debemos “reinventar la rueda” en varias de nuestras aplicaciones. Hay mucho trabajo ya realizado, testeado y aceptado que en un entorno similar a mi problema ya aporta una solución satisfactoria. ¿Para qué voy a inventar un ladrillo si ya otro lo hizo y el mismo ya fue usado en la edificación de millones de estructuras con éxito?

## actividad

1. Realizar un ejemplo de los patrones creacionales.

* El ejemplo lo pueden descargar de este [**link**](http://www.fluffycat.com/PHP-Design-Patterns/).
* Deben crear dentro de la carpeta del ejercicio un subdirectorio con el nombre de patrón de diseño y los archivos que se generaron.

1. Realizar un ejemplo de los patrones estructurales.

* El ejemplo lo pueden descargar de este [**link**](http://www.fluffycat.com/PHP-Design-Patterns/).
* Deben crear dentro de la carpeta del ejercicio un subdirectorio con el nombre de patrón de diseño y los archivos que se generaron.

1. Realizar un ejemplo de los patrones de comportamiento.

* El ejemplo lo pueden descargar de este [**link**](http://www.fluffycat.com/PHP-Design-Patterns/).
* Deben crear dentro de la carpeta del ejercicio un subdirectorio con el nombre de patrón de diseño y los archivos que se generaron.

**NOTA**:

* El nombre de la carpeta general donde estará contenido todo el archivo del tema se llama **ejercicio04**.
* La carpeta se enviará por correo electrónico como archivo **ZIP** con el nombre de DAII-Ejercicio04-NombreApellido.zip y el asunto de correo debe ser el mismo nada más sin el tipo de archivo **ZIP**.
* Al finalizar la clase sesión de clases se deberán enviar los tres productos por correo electrónico, con un tiempo de tolerancia máximo de 15 minutos.

## Rubrica.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Puntos | |  |
| Aspecto | 10 | 0 | Puntos |
| * 1. Existe el folder del patrón de diseño creacionales y sus archivos. | Todos los archivos | Faltan archivos |  |
| * 1. Se ejecuta el ejercicio satisfactoriamente del patrón creacional. | Completo y sin error. | Incompleto o sin error |  |
| * 1. Existe el folder del patrón de diseño estructurales y sus archivos. | Todos los archivos | Faltan archivos |  |
| * 1. Se ejecuta el ejercicio satisfactoriamente del patrón estructural. | Completo y sin error. | Incompleto o sin error |  |
| * 1. Existe el folder del patrón de diseño de comportamiento y sus archivos. | Todos los archivos | Faltan archivos |  |
| * 1. Se ejecuta el ejercicio satisfactoriamente del patrón de comportamiento. | Completo y sin error. | Incompleto o sin error |  |

## material de consulta

* <http://siul02.si.ehu.es/~alfredo/iso/06Patrones.pdf>
* <https://platzi.com/blog/patrones-de-diseno/>
* <http://www.lsi.us.es/docencia/get.php?id=436>