Patrons de disseny

Els patrons de disseny són solucions típiques als problemes que es produeixen habitualment en el disseny de programari.

No podeu trobar un patró i copiar-lo al vostre programa, de la manera que podeu fer amb les funcions o biblioteques disponibles.

El patró no és una peça específica de codi, sinó un concepte general per resoldre un problema particular. Podeu seguir els detalls del patró i implementar una solució que s'adapti a les realitats del vostre propi programa.

Els patrons de disseny es diferencien per la seva complexitat, nivell de detall i escala d'aplicabilitat a tot el sistema que s'està dissenyant.

Classificació dels patrons

Els patrons es poden classificar segons la seva intenció o propòsit.

**Patrons de creació (Creational patterns)**

Els patrons de creació proporcionen mecanismes de creació d'objectes que augmenten la flexibilitat i la reutilització del codi existent. Dins d’aquest tipus de patró podem trobar:

* **Mètode de fàbrica (Factory Method)** és un patró de disseny de creació que proporciona una interfície per crear objectes en una superclasse, però permet que les subclasses alterin el tipus d'objectes que es crearan.
* **Fàbrica abstracta (Abstract Factory)** és un patró de disseny creatiu que us permet produir famílies d'objectes relacionats sense especificar les seves classes concretes.
* **Constructor (Builder)** és un patró de disseny creatiu que us permet construir objectes complexos pas a pas. El patró permet produir diferents tipus i representacions d'un objecte utilitzant el mateix codi de construcció.
* El **Prototip** és un patró de disseny de creació que us permet copiar objectes existents sense fer que el vostre codi depengui de les seves classes.
* El **Singleton** és un patró de disseny de creació que us permet assegurar-vos que una classe només té una instància, alhora que proporciona un punt d'accés global a aquesta instància.

**Patrons estructurals (Structural patterns)**

Els patrons estructurals expliquen com muntar objectes i classes en estructures més grans, alhora que mantenen aquestes estructures flexibles i eficients. Dins d’aquest tipus de patró podem trobar:

* L’ **Adaptador** és un patró de disseny estructural que permet col·laborar objectes amb interfícies incompatibles.
* El **Pont (Bridge)** és un patró de disseny estructural que us permet dividir una classe gran o un conjunt de classes estretament relacionades en dues jerarquies separades (abstracció i implementació) que es poden desenvolupar independentment l'una de l'altra.
* **Composite** és un patró de disseny estructural que us permet compondre objectes en estructures d'arbre i després treballar amb aquestes estructures com si fossin objectes individuals.
* **Decorator** és un patró de disseny estructural que us permet adjuntar nous comportaments als objectes col·locant aquests objectes dins d'objectes d'embolcall especials que contenen els comportaments.
* **Facade** és un patró de disseny estructural que proporciona una interfície simplificada a una biblioteca, un marc o qualsevol altre conjunt complex de classes.
* **Pes Mosca (Flyweight)** és un patró de disseny estructural que us permet encaixar més objectes a la quantitat de memòria RAM disponible compartint parts comunes d'estat entre diversos objectes en lloc de mantenir totes les dades de cada objecte.
* El **Proxy** és un patró de disseny estructural que us permet proporcionar un substitut o un marcador de posició per a un altre objecte. Un servidor intermediari controla l'accés a l'objecte original, cosa que us permet fer alguna cosa abans o després que la sol·licitud arribi a l'objecte original.

**Patrons de comportament (Behavioral patterns)**

Els patrons de comportament tenen cura de la comunicació eficaç i de l'assignació de responsabilitats entre objectes. Dins d’aquest tipus de patró podem trobar:

* **Chain of Responsibility** és un patró de disseny de comportament que et permet passar sol·licituds al llarg d'una cadena de manejadors. En rebre una sol·licitud, cada manejador decideix si la processa o si la passa al següent manejador de la cadena.
* **Command** és un patró de disseny de comportament que converteix una sol·licitud en un objecte independent que conté tota la informació sobre la sol·licitud. Aquesta transformació permet parametritzar els mètodes amb diferents sol·licituds, retardar o posar en cua l'execució d'una sol·licitud i suportar operacions que no es poden fer.
* **Iterator** és un patró de disseny de comportament que et permet recórrer elements d'una col·lecció sense exposar-ne la representació subjacent (llista, pila, arbre, etc.).
* **Mediator** és un patró de disseny de comportament que permet reduir les dependències caòtiques entre objectes. El patró restringeix les comunicacions directes entre els objectes i els força a col·laborar únicament a través d'un objecte mediador.
* **Memento** és un patró de disseny de comportament que us permet guardar i restaurar l'estat previ d'un objecte sense revelar els detalls de la seva implementació.
* **Observer** és un patró de disseny de comportament que et permet definir un mecanisme de subscripció per notificar a diversos objectes sobre qualsevol esdeveniment que li succeeix a fi que estan observant.
* **State** és un patró de disseny de comportament que permet a un objecte alterar-ne el comportament quan el seu estat intern canvia. Sembla com si l'objecte canviés la classe.
* **Strategy** és un patró de disseny de comportament que et permet definir una família d'algorismes, col·locar-los cadascun en una classe separada i fer els seus objectes intercanviables.
* **Template Method** és un patró de disseny de comportament que defineix l'esquelet d'un algorisme a la superclasse però permet que les subclasses sobreescriguin passos de l'algoritme sense canviar-ne l'estructura.
* **Visitor** és un patró de disseny de comportament que permet separar algorismes dels objectes sobre els quals operen.