**PATRONES DE DISEÑO**

**Abstract Factory:** tiene como objetivo la creación de objetos reagrupados en familias sin tener que conocer las clases concretas destinadas a la creación de estos objetos;

**Builder:** permite separar la construcción de objetos complejos de su implementación de modo que un cliente pueda crear estos objetos complejos con implementaciones diferentes;

**Factory Method:** tiene como objetivo presentar un método abstracto para la creación de un objeto reportando a las subclases concretas la creación efectiva;

**Prototype:** permite crear nuevos objetos por duplicación de objetos existentes llamados prototipos que disponen de la capacidad de clonación;

**Singleton:** permite asegurar que de una clase concreta existe una única instancia y proporciona un método único que la devuelve;

**Adapter:** tiene como objetivo convertir la interfaz de una clase existente en la interfaz esperada por los clientes también existentes para que puedan trabajar de forma conjunta;

**Bridge:** tiene como objetivo separar los aspectos conceptuales de una jerarquía de clases de su implementación;

**Composite:** proporciona un marco de diseño de una composición de objetos con una profundidad de composición variable, basando el diseño en un árbol;

**Decorator:** permite agregar dinámicamente funcionalidades suplementarias a un objeto;

**Facade:** tiene como objetivo reagrupar las interfaces de un conjunto de objetos en una interfaz unificada que resulte más fácil de utilizar;

**Flyweight:** facilita la compartición de un conjunto importante de objetos con granularidad muy fina;

**Proxy:** construye un objeto que se substituye por otro objeto y que controla su acceso;

**Chain of responsibility:** crea una cadena de objetos tal que si un objeto de la cadena no puede responder a una petición, la pueda transmitir a sus sucesores hasta que uno de ellos responda;

**Command:** tiene como objetivo transformar una consulta en un objeto, facilitando operaciones como la anulación, la actualización de consultas y su seguimiento;

**Interpreter:** proporciona un marco para dar una representación mediante objetos de la gramática de un lenguaje con el objetivo de evaluar, interpretándolas, expresiones escritas en este lenguaje;

**Iterator:** proporciona un acceso secuencial a una colección de objetos sin que los clientes se preocupen de la implementación de esta colección;

**Mediator:** construye un objeto cuya vocación es la gestión y el control de las interacciones en el seno de un conjunto de objetos sin que estos elementos se conozcan mutuamente;

**Memento:** salvaguarda y restaura el estado de un objeto;

**Observer:** construye una dependencia entre un sujeto y sus observadores de modo que cada modificación del sujeto sea notificada a los observadores para que puedan actualizar su estado;

**State:** permite a un objeto adaptar su comportamiento en función de su estado interno;

**Strategy:** adapta el comportamiento y los algoritmos de un objeto en función de una necesidad concreta sin por ello cargar las interacciones con los clientes de este objeto;

**Template Method:** permite reportar en las subclases ciertas etapas de una de las operaciones de un objeto, estando éstas descritas en las subclases;

**Visitor:** construye una operación a realizar en los elementos de un conjunto de objetos. Es posible agregar nuevas operaciones sin modificar las clases de estos objetos.