* PHP design patterns
  + Design patterns are necessary to understand core of most of the frameworks have code written using various design patterns. To understant the basics
  + Architectural pattern
    - Even though an architectural pattern conveys an image of a system, it is not an architecture. An architectural pattern is a concept that solves and delineates some essential cohesive elements of a software architecture. Countless different architectures may implement the same pattern and share the related characteristics. Patterns are often defined as "strictly described and commonly available
    - An architectural style defines: a family of systems in terms of a pattern of structural organization; a vocabulary of components and connectors, with constraints on how they can be combined
    - Data integration/SOA
      * Extract, transform, load
        + in computing, extract, transform, load (ETL) is a three-phase process where data is extracted, transformed (cleaned, sanitized, scrubbed) and loaded into an output data container. The data can be collated from one or more sources and it can also be outputted to one or more destinations.
      * Managed file transfer
        + Managed file transfer (MFT) refers to a technology that provides the secure transfer of data in an efficient and reliable manner. MFT software is marketed to companies as a more secure alternative to using insecure protocols like FTP (file transfer protocol) and HTTP to transfer files
      * Enterprise service bus
        + An enterprise service bus (ESB) implements a communication system between mutually interacting software applications in a service-oriented architecture (SOA). It represents a software architecture for distributed computing, and is a special variant of the more general client-server model, wherein any application may behave as server or client
      * Data architecture
        + Data architecture consist of models, policies, rules, and standards that govern which data is collected and how it is stored, arranged, integrated, and put to use in data systems and in organizations.
      * Data modelling
        + Data modeling in software engineering is the process of creating a data model for an information system by applying certain formal techniques.
        + Dimensional modelling

Dimensional modeling (DM) is part of the Business Dimensional Lifecycle methodology developed by Ralph Kimball which includes a set of methods, techniques and concepts for use in data warehouse design

* + - * + E-R data modelling

An entity–relationship model (or ER model) describes interrelated things of interest in a specific domain of knowledge.

* + - * Artificial intelligence
        + Decision management
        + Text analytics and NLP
        + Classic machine learning
      * the blackboard pattern is a behavioral design pattern that provides a computational framework for the design and implementation of systems that integrate large and diverse specialized modules, and implement complex, non-deterministic control strategies.
      * The broker pattern is an architectural pattern that can be used to structure distributed software systems with decoupled components that interact by remote procedure calls. A broker component is responsible for coordinating communication, such as forwarding requests, as well as for transmitting results and exceptions.
      * In software object-oriented design, a layer is a group of classes that have the same set of
      * link-time module dependencies to other modules
      * Action–domain–responder (ADR) is a software architectural pattern that was proposed by Paul M. Jones as a refinement of Model–view–controller (MVC) that is better suited for web applications
      * Model–view–controller (MVC) is a software architectural pattern commonly used for developing user interfaces that divide the related program logic into three interconnected elements.
        + Model

The central component of the pattern. It is the application's dynamic data structure, independent of the user interface

It directly manages the data, logic and rules of the application.

* + - * + View

Any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

a view is just a visual representation of a model, and does not handle user input

view represents a complete user interface element such as a menu or button, and does receive input from the user

* + - * + Controller

Accepts input and converts it to commands for the model or view

controller handles user input events, such as button presses or mouse movement At any given time, each controller has one associated view and model, although one model object may hear from many different controllers.

* + - * Presentation–abstraction–control
        + It is an interaction-oriented software architecture, and is somewhat similar to model–view–controller (MVC) in that it separates an interactive system into three types of components responsible for specific aspects of the application's functionality
        + The abstraction component retrieves and processes the data, the presentation component formats the visual and audio presentation of data, and the control component handles things such as the flow of control and communication between the other two components
        + Model–view–viewmodel (MVVM) is an architectural pattern in computer software that facilitates the separation of the development of the graphical user interface (GUI; the view)—be it via a markup language or GUI code—from the development of the business logic or back-end logic (the model) such that the view is not dependent upon any specific model platform.
      * Object-oriented programming
        + Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects",Which can contain data and code
        + The data is in the form of fields (often known as attributes or properties), and the code is in the form of procedures
        + A common feature of objects is that procedures (or methods) are attached to them and can access and modify the object's data fields. In this brand of OOP, there is usually a special name such as this or self used to refer to the current object.
  + Software design pattern
    - In software engineering, a software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design.
    - Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.
    - Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved
    - In databases, change data capture (CDC) is a set of software design patterns used to determine and track the data that has changed so that action can be taken using the changed data.
    - Data mining is the process of extracting and discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems
    - Data modelling
      * Modeling standards
      * Naming conventions
    - Effective software design requires considering issues that may not become visible until later in the implementation
    - Software design techniques are difficult to apply to a broader range of problems
    - Design patterns provide general solutions, Structure
    - Domain-specific patterns
      * Efforts have also been made to codify design patterns in particular domains, including use of existing design patterns as well as domain-specific design patterns
    - Classification and list
      * Design patterns had originally been categorized into 3 sub-classifications based on what kind of problem they solve.
      * Creational patterns
        + provide the capability to create objects based on a required criterion and in a controlled way.
        + Abstract factory

Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Use Design and code Patterns

* + - * + Builder

Separate the construction of a complex object from its representation, allowing the same construction process to create various representations

Only use in Design Patterns

* + - * + Dependency Injection

In software engineering, dependency injection is a design pattern in which an object or function receives other objects or functions that it depends on. A form of inversion of control, dependency injection aims to separate the concerns of constructing objects and using them, leading to loosely coupled programs

Types of dependency injection

**Constructor** **injection**, where dependencies are provided through a client's class constructor.

**Setter injection**, where the client exposes a setter method which accepts the dependency.

**Interface injection,** where the dependency's interface provides an injector method that will inject the dependency into any client passed to it

A class accepts the objects it requires from an injector instead of creating the objects directly.

Not Use Design and code Patterns

* + - * + Factory method

In class-based programming, the factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This is done by creating objects by calling a factory method

Define an interface for creating a single object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

Use Design and code Patterns

* + - * + Lazy initialization

Tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is needed. This pattern appears in the GoF catalog as "virtual proxy", an implementation strategy for the Proxy pattern.

Not Use Design and code Patterns

* + - * + Multiton

Ensure a class has only named instances, and provide a global point of access to them

Not Use Design and code Patterns

* + - * + Object pool

Avoid expensive acquisition and release of resources by recycling objects that are no longer in use. Can be considered a generalisation of connection pool and thread pool patterns

Not Use Design and code Patterns

* + - * + Prototype

Specify the kinds of objects to create using a prototypical instance, and create new objects from the 'skeleton' of an existing object, thus boosting performance and keeping memory footprints to a minimum.

To implement the pattern, the client declares an abstract base class that specifies a pure virtual clone() method. Any class that needs a "polymorphic constructor" capability derives itself from the abstract base class, and implements the clone() operation

Use Design Patterns

* + - * + Resource acquisition is initialization

Ensure that resources are properly released by tying them to the lifespan of suitable objects.

Not Use Design and code Patterns

* + - * + Singleton

Ensure a class has only one instance, and provide a global point of access to it.

Use Design and code Patterns

* + - * Structural patterns
        + Are about organizing different classes and objects to form larger structures and provide new functionality.
        + Adapter, Wrapper, or Translator

In software engineering, the adapter pattern is a software design pattern (also known as wrapper, an alternative naming shared with the decorator pattern) that allows the interface of an existing class to be used as another interface. It is often used to make existing classes work with others without modifying their source code.

Convert the interface of a class into another interface clients expect. An adapter lets classes work together that could not otherwise because of incompatible interfaces. The enterprise integration pattern equivalent is the translator.

Use Design and code Patterns

* + - * + Bridge

Decouple an abstraction from its implementation allowing the two to vary independently.

Use Design and code Patterns

* + - * + Composite

Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Use Design and code Patterns

* + - * + Decorator

In object-oriented programming, the decorator pattern is a design pattern that allows behavior to be added to an individual object, dynamically, without affecting the behavior of other objects from the same class

Attach additional responsibilities to an object dynamically keeping the same interface. Decorators provide a flexible alternative to subclassing for extending functionality.

Use Design and code patterns

* + - * + Extension object

Adding functionality to a hierarchy without changing the hierarchy.

Not Use Design and code patterns

* + - * + Facade

The facade pattern (also spelled façade) is a software-design pattern commonly used in object-oriented programming. Analogous to a facade in architecture, a facade is an object that serves as a front-facing interface masking more complex underlying or structural code

improve the readability and usability of a software library by masking interaction with more complex components behind a single

provide a context-specific interface to more generic functionality (complete with context-specific input validation)

Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

Not Use Design and code patterns

* + - * + Flyweight

Use sharing to support large numbers of similar objects efficiently.

Use Design

* + - * + Front controller

The pattern relates to the design of Web applications. It provides a centralized entry point for handling requests.

Not Use Design and code patterns

* + - * + Marker

Empty interface to associate metadata with a class.

Not Use Design and code patterns

* + - * + Module

Group several related elements, such as classes, singletons, methods, globally used, into a single conceptual entity.

* + - * + Proxy

Provide a surrogate or placeholder for another object to control access to it.

A proxy, in its most general form, is a class functioning as an interface to something else. The proxy could interface to anything: a network connection, a large object in memory, a file, or some other resource that is expensive or impossible to duplicate. In short, a proxy is a wrapper or agent object that is being called by the client to access the real serving object behind the scenes. Use of the proxy can simply be forwarding to the real object, or can provide additional logic.

Use Design

* + - * + Twin

Twin allows modeling of multiple inheritance in programming languages that do not support this feature.

Not Use Design and code patterns

* + - * behavioral patterns
        + are about identifying common communication patterns between objects and realizing these patterns.
        + Blackboard

In software engineering, the blackboard pattern is a behavioral design pattern that provides a computational framework for the design and implementation of systems that integrate large and diverse specialized modules, and implement complex, non-deterministic control strategies

Not Use Design and code patterns

* + - * + Chain of responsibility

Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

Use Design

* + - * + Command

Encapsulate a request as an object, thereby allowing for the parameterization of clients with different requests, and the queuing or logging of requests. It also allows for the support of undoable operations.

Use Design

* + - * + Interpreter

Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

Use Design

* + - * + Iterator

In object-oriented programming, the iterator pattern is a design pattern in which an iterator is used to traverse a container and access the container's elements

Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Use Design

* + - * + Mediator

Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it allows their interaction to vary independently.

Use Design

* + - * + Memento

Without violating encapsulation, capture and externalize an object's internal state allowing the object to be restored to this state later.

Use Design

* + - * + Null object

Avoid null references by providing a default object.

Not use Use Design and code

* + - * + Observer or Publish/subscribe

Define a one-to-many dependency between objects where a state change in one object results in all its dependents being notified and updated automatically.

Use Design and code

* + - * + Servant

Define common functionality for a group of classes. The servant pattern is also frequently called helper class or utility class implementation for a given set of classes. The helper classes generally have no objects hence they have all static methods that act upon different kinds of class objects.

Not Use Design and code

* + - * + Specification

Recombinable business logic in a Boolean fashion.

Not Use Design and code

* + - * + State

Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

Use Design

* + - * + Strategy

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Use Design and code—from

* + - * + Template method

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Use Design and code

* + - * + Visitor

Represent an operation to be performed on instances of a set of classes. Visitor lets a new operation be defined without changing the classes of the elements on which it operates.

Use Design

* + - * + Fluent Interface

Design an API to be method chained so that it reads like a DSL. Each method call returns a context through which the next logical method call(s) are made available.

Not Use Design and code

* + - * + Active Object

Decouples method execution from method invocation that reside in their own thread of control. The goal is to introduce concurrency, by using asynchronous method invocation and a scheduler for handling requests.

* + - * + Balking

Only execute an action on an object when the object is in a particular state.

* + - * + Binding properties

Combining multiple observers to force properties in different objects to be synchronized or coordinated in some way

* + - * + Compute kernel

The same calculation many times in parallel, differing by integer parameters used with non-branching pointer math into shared arrays, such as GPU-optimized Matrix multiplication or Convolutional neural network.

* + - * + Double-checked locking

Reduce the overhead of acquiring a lock by first testing the locking criterion (the 'lock hint') in an unsafe manner; only if that succeeds does the actual locking logic proceed.

Can be unsafe when implemented in some language/hardware combinations. It can therefore sometimes be considered an anti-pattern.

* + - * + Event-based asynchronous

Addresses problems with the asynchronous pattern that occur in multithreaded programs.

* + - * + Guarded suspension

Manages operations that require both a lock to be acquired and a precondition to be satisfied before the operation can be executed.

* + - * + Join

Join-pattern provides a way to write concurrent, parallel and distributed programs by message passing. Compared to the use of threads and locks, this is a high-level programming model.\\

* + - * + Lock

One thread puts a "lock" on a resource, preventing other threads from accessing or modifying it

* + - * + Messaging design pattern (MDP)

Allows the interchange of information (i.e. messages) between components and applications.

* + - * + Monitor object

An object whose methods are subject to mutual exclusion, thus preventing multiple objects from erroneously trying to use it at the same time.

* + - * + Reactor

A reactor object provides an asynchronous interface to resources that must be handled synchronously.

* + - * + Read-write lock

Allows concurrent read access to an object, but requires exclusive access for write operations. An underlying semaphore might be used for writing, and a Copy-on-write mechanism may or may not be used.

* + - * + Scheduler

Explicitly control when threads may execute single-threaded code.

* + - * + Thread pool

A number of threads are created to perform a number of tasks, which are usually organized in a queue. Typically, there are many more tasks than threads. Can be considered a special case of the object pool pattern.

* + - * + Thread-specific storage

Static or "global" memory local to a thread.

Safe Concurrency with Exclusive Ownership

Avoiding the need for runtime concurrent mechanisms, because exclusive ownership can be proven. This is a notable capability of the Rust language, but compile-time checking isn't the only means, a programmer will often manually design such patterns into code - omitting the use of locking mechanism because the programmer assesses that a given variable is never going to be concurrently accessed.

* + - * + CPU atomic operation
  + Collection of implementation
    - This is a collection of known design patterns and some sample code how to implement them in PHP. Every pattern has a small list of examples.
    - Patterns
      * Creational
        + In software engineering, creational design patterns are design patterns that deal with object creation mechanisms
        + trying to create objects in a manner suitable to the situation.
        + The basic form of object creation could result in design problems or added complexity to the design.
        + Abstract Factory

To create series of related or dependent objects without specifying their concrete classes. Usually the created classes all implement the same interface. The client of the abstract factory does not care about how these objects are created, it just knows how they go together

>composer require --dev phpunit/phpunit ^7