Design Patterns in PHP

Introduction

What are design patterns?

- Solutions to recurring problems
- Guidelines on how to tackle certain problems
- Not a silver bullet to all your problems
- Do not try to force them
- Are not solutions looking for problems

Types of design patterns

- Creational Design Patterns
- Structural Design Patterns
- Behavioral Design Patterns

Creational Design Patterns

Creational patterns are focused towards how to instantiate an object or group of related objects.

Creational Design Patterns

- ☝ Singleton
- 🏠 Simple Factory
- 🏭 Factory Method
- 🔨 Abstract Factory
- 👷 Builder
- 🐑 Prototype

Structural Design Patterns

Structural patterns are mostly concerned with object composition or in other words how the entities can use each other.

Structural Design Patterns

- 🔌 Adapter
- 🌉 Bridge
- 🌿 Composite
- ☕ Decorator
- 🗃️ Facade
- 💦 Flyweight
- 🔐 Proxy

Behavioral Design Patterns

Behavioral design patterns are design patterns that identify common communication patterns between objects and realize these patterns.

Behavioral Design Patterns

- ⛓️ Chain of Responsibility
- 👮 Command
- 💤 Iterator
- 📩 Mediator
- 🔙 Memento
- 🐵 Observer
- 🏃 Visitor
- 🤲 Strategy
- 💢 State
- 📙 Template Method

Singleton Design pattern

Real world Example

There can only be one president of a country at a time. The same president has to be brought to action, whenever duty calls. President here is singleton.

In plain words

Ensures that only one object of a particular class is ever created.

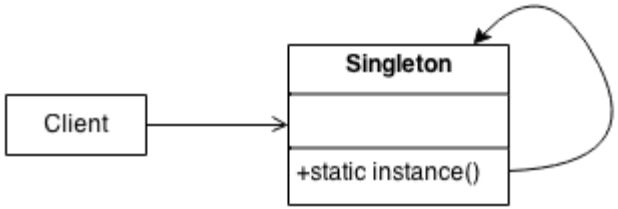
Wikipedia says

In software engineering, the singleton pattern is a software design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system.

Code Example

https://github.com/Bootsity/design-patterns-php/tree/master/Creational/singleton

Generalized UML



Simple Factory Pattern

Real world Example

Consider, you are building a house and you need doors. It would be a mess if every time you need a door, you put on your carpenter clothes and start making a door in your house. Instead you get it made from a factory.

In plain words

Simple factory simply generates an instance for client without exposing any instantiation logic to the client

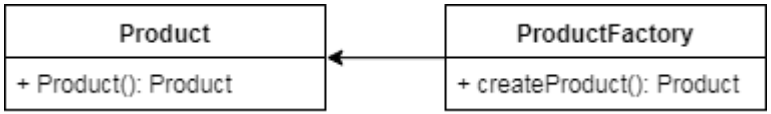
Wikipedia says

In object-oriented programming (OOP), a factory is an object for creating other objects – formally a factory is a function or method that returns objects of a varying prototype or class from some method call, which is assumed to be "new".

Code Example

https://github.com/Bootsity/design-patterns-php/tree/master/Creational/simpleFactory

Generalized UML



Factory Method Pattern

Real world Example

Consider the case of a hiring manager. It is impossible for one person to interview for each of the positions. Based on the job opening, she has to decide and delegate the interview steps to different people.

In plain words

It provides a way to delegate the instantiation logic to child classes.

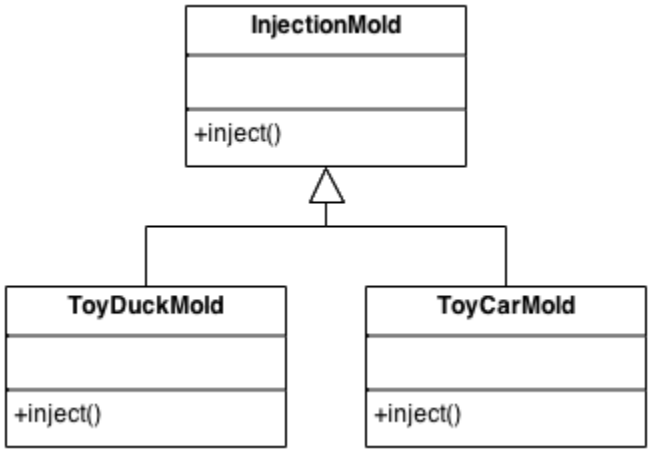
Wikipedia says

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—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.

Code Example

https://github.com/Bootsity/design-patterns-php/tree/master/Creational/factoryMethod

Generalized UML



Abstract Factory Pattern

Real world Example

Extending our door example from Simple Factory. Based on your needs you might get a wooden door from a wooden door shop, iron door from an iron shop or a PVC door from the relevant shop. Plus you might need a guy with different kind of specialties to fit the door, for example a carpenter for wooden door, welder for iron door etc. As you can see there is a dependency between the doors now, wooden door needs carpenter, iron door needs a welder etc.

In plain words

A factory of factories; a factory that groups the individual but related/dependent factories together without specifying their concrete classes.

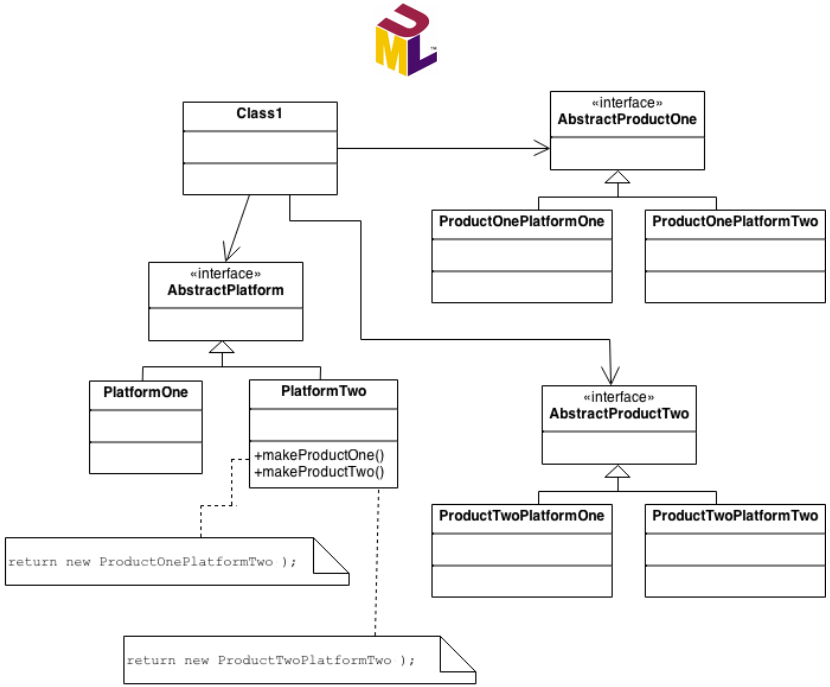
Wikipedia says

The abstract factory pattern provides a way to encapsulate a group of individual factories that have a common theme without specifying their concrete classes

Code Example

https://github.com/Bootsity/design-patterns-php/tree/master/Creational/abstractfactory

Generalized UML



Builder Pattern

Real world Example

Imagine you are at Hardee's and you order a specific deal, lets say, "Big Hardee" and they hand it over to you without any questions; this is the example of simple factory. But there are cases when the creation logic might involve more steps. For example you want a customized Subway deal, you have several options in how your burger is made e.g what bread do you want? what types of sauces would you like? What cheese would you want? etc. In such cases builder pattern comes to the rescue.

In plain words

Allows you to create different flavors of an object while avoiding constructor pollution. Useful when there could be several flavors of an object. Or when there are a lot of steps involved in creation of an object.

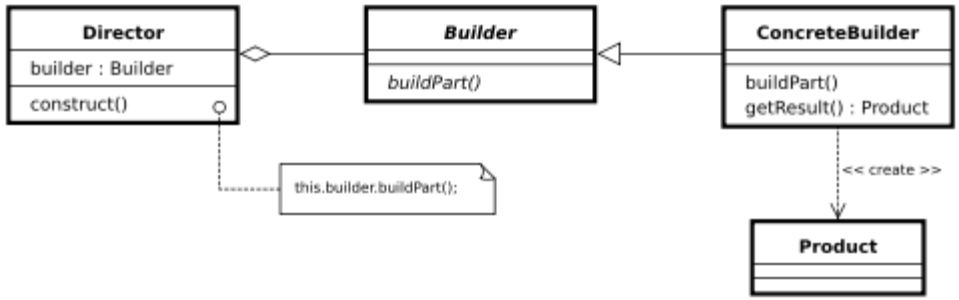
Wikipedia says

The builder pattern is an object creation software design pattern with the intentions of finding a solution to the telescoping constructor anti-pattern.

Code Example

https://github.com/Bootsity/design-patterns-php/tree/master/Creational/builder

Generalized UML



Prototype Pattern

Real world Example

Remember dolly? The sheep that was cloned! Lets not get into the details but the key point here is that it is all about cloning

In plain words

Create object based on an existing object through cloning.

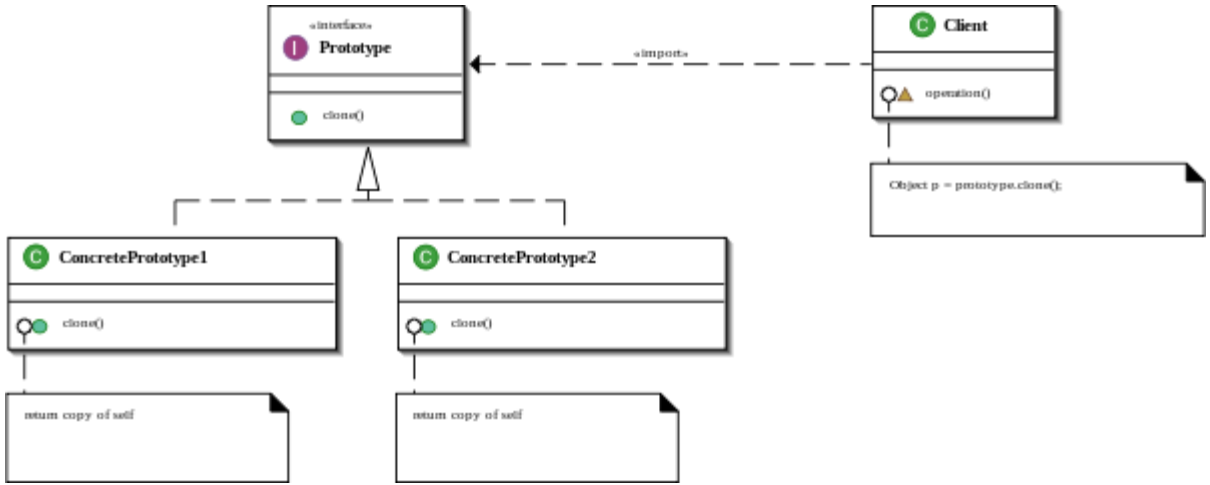
Wikipedia says

The prototype pattern is a creational design pattern in software development. It is used when the type of objects to create is determined by a prototypical instance, which is cloned to produce new objects.

Code Example

https://github.com/Bootsity/design-patterns-php/tree/master/Creational/prototype

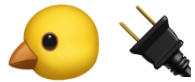
Generalized UML



Adapter Pattern

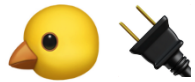
Real world Example

In order to transfer pictures from your camera’s memory card to your computer, you need some kind of adapter that is compatible with your computer ports so that you can attach memory card to your computer. In this case card reader is an adapter.



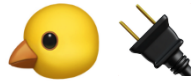
In plain words

Adapter pattern lets you wrap an otherwise incompatible object in an adapter to make it compatible with another class.



Wikipedia says

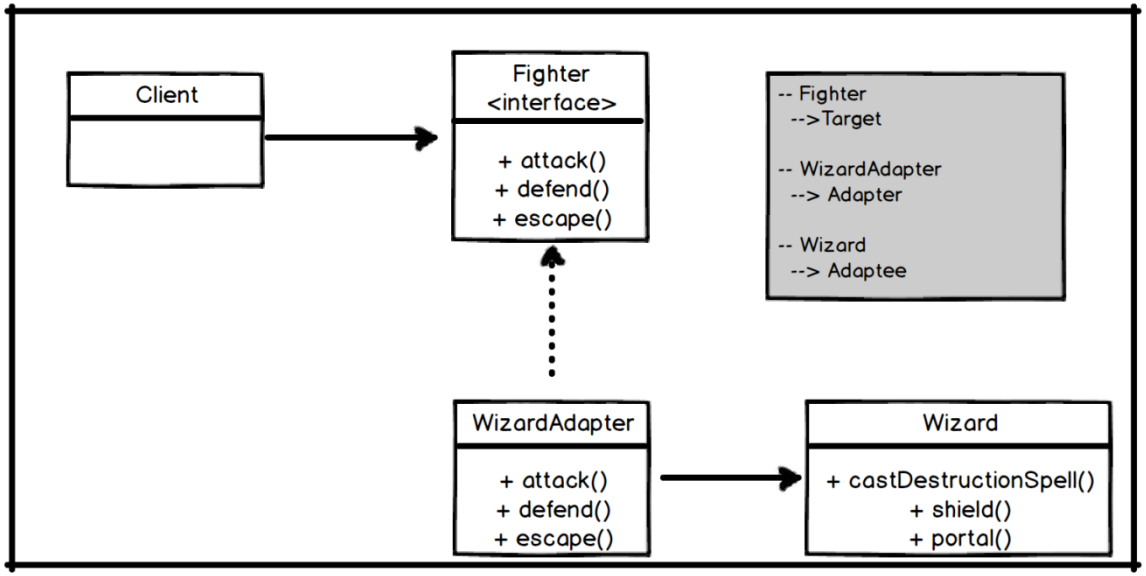
In software engineering, the adapter pattern is a software design pattern that allows the interface of an existing class to be used as another interface. It is often used to make existing classes work with others without modifying their source code.



Code Example

https://github.com/Bootsity/design-patterns-php/tree/master/Structural/adapter

Generalized UML



Bridge Pattern

Real world Example

Consider you have a website with different pages and you are supposed to allow the user to change the theme. What would you do? Create multiple copies of each of the pages for each of the themes or would you just create separate theme and load them based on the user's preferences? Bridge pattern allows you to do the second i.e.

In plain words

Bridge pattern is about preferring composition over inheritance. Implementation details are pushed from a hierarchy to another object with a separate hierarchy.

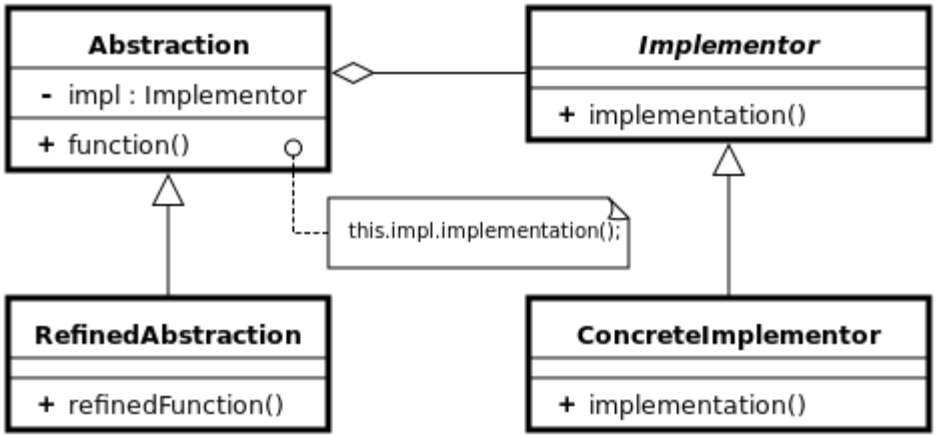
Wikipedia says

The bridge pattern is a design pattern used in software engineering that is meant to "decouple an abstraction from its implementation so that the two can vary independently".

Code Example

https://github.com/Bootsity/design-patterns-php/tree/master/Structural/bridge

Generalized UML



Composite Pattern

Real world Example

Every organization is composed of employees. Each of the employees has the same features i.e. has a salary, has some responsibilities, may or may not report to someone, may or may not have some subordinates etc.

In plain words

Composite pattern lets clients treat the individual objects in a uniform manner.

Wikipedia says

In software engineering, the composite pattern is a partitioning design pattern. The composite pattern describes that a group of objects is to be treated in the same way as a single instance of an object. The intent of a composite is to "compose" objects into tree structures to represent part-whole hierarchies. Implementing the composite pattern lets clients treat individual objects and compositions uniformly.

Code Example

https://github.com/Bootsity/design-patterns-php/tree/master/Structural/composite

Generalized UML