Scale

* An algorithm, design, networking protocol, program, or other system is said to scale if it is suitably efficient and practical when applied to large situations (e.g. a large input data set, a large number of outputs or users, or a large number of participating nodes in the case of a distributed system). If the design or system fails when a quantity increases, it does not scale. In practice, if there are a large number of things (n) that affect scaling, then resource requirements (for example, algorithmic time-complexity) must grow less than n2 as n increases. An example is a search engine that must scale not only for the number of users but for the number of objects it indexes. Scalability refers to the ability of a site to increase in size as demand warrants.

Why use OOP

* Organize large projects into manageable pieces
* Procedural:
* Step by step instructions that compute must follow
* A procedural program can consist of a series of conditions and function calls for logic, but the end result is very linear.
* There is no intrinsically wrong with procedural programming and for small projects procedural programming is can be appropriate practice.
* The problems with scale up.
* We may find us defining arrays with long descriptive keys, or passing large numbers of global variables between scripts, or searching for a nonfatal bug across a dozen scripts with thousands of lines of code with little indication of which component is at fault.
* With that I mind, object-oriented programming introduces structure and principles aimed at mitigating procedural scaling and maintainability issues.
* Object-oriented programming features
* Abstraction
* Defines data and program structures using a representation of the meaning while hiding the implementation itself.
* Hides implementation itself
* Encapsulation
* Exposes functionality, restricts access to low-level components and data
* Hierarchy (or Inheritance)
* Inherited attributes and behavior from pre-existing classes are inherited, allowing for incremental development; modularity.
* Incremental development
* Modularity – functionality is broken into task specific pieces
* Functionality is broken into modules that accomplish one task and contain everything to complete said task
* Instead of trying to deal with one large problem, a number of smaller sub problems work in conjunction with each other
* Polymorphism
* Interact with classes without knowing which class it is.
* By implementing this technique, the compartmentalized components can be maintained and expanded without affecting the overall architecture, which in turn reduces overhead.
* Polymorphism allows values of different data types to be handled using a uniform interface.

Examples

Characters

- HP

- ArmorValue

- MagicPower

Wolf extend characters

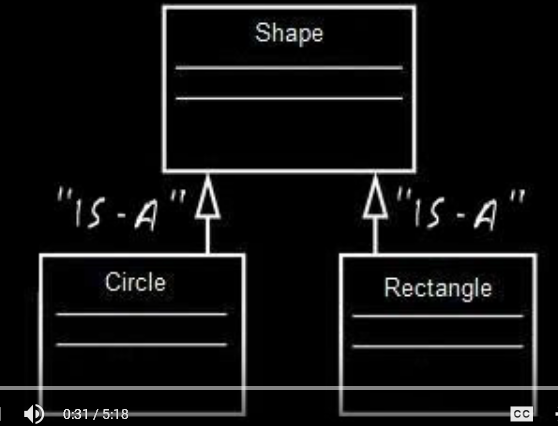
Horse extend characters

Dragon extend characters

Human extend characters

Polymorphism & Inheritance

* When an objects derived from common base class then we said there is a “IS-A” relationship between derived and base classes.
* Example:



* Here base class shape can’t be instantiated because it’s an abstract class.
* Shape will provide common features all of the sub class.

My thought about abstraction

* Gives the common feature to the sub class
* It likes as template tell the sub class implement as your wish
* Never need to instantiate the abstract class
* If we declare anything in abstract class that means it should be implemented in sub class.
* We can’t declare properties as abstract in parent class.

Benefits of OOP:

* Create manageable projects more easily.
* OOP provides facilities to create reusable objects
* Almost all big applications are written using the object oriented approach.
* Reusability: Prevent DRY (Don’t Repeat Yourself).
* Refactoring: All objects are small entities and contain its properties and methods as a part of itself. So refactoring is comparatively easier.
* Extensible: We can extend the object and create a totally new object that retains all the necessary properties and methods of the parent object from which it has been derived, and then expose new features.
* Maintenance: Very easy to merge the code.
* Efficiency: The concept of object oriented programming is actually developed for better efficiency and ease of development process. Because you first split your problem into a small set of problems and then find solutions for each of them, the big problem is solved automatically.

Class:

* Blueprint of the object.
* A class is a template for an object.
* A class may contain its own constants, variables (called "properties"), and functions (called "methods").
* The pseudo-variable $this is available when a method is called from within an object context. $this is a reference to the calling object.

Object:

* Code grouped by common theme.
* Abstractions of code structures.
* An object can be almost any item or concept—a physical object such as a desk or a customer; or a conceptual object that exists only in software, such as a text input area or a file.
* A piece of code with a bunch of properties and methods.
* Objects are basically small discrete pieces of code which, can incorporate data and behaviors together.
* In an application all these objects are connected to each other, they share data among them and solve problems.
* Every object has some attributes and operations.
* Attributes are properties or variables or fields that relate to the object.
* Non-static properties may be accessed by using -> like: $this->property
* Static properties are accessed by using the :: (Double Colon): self::$property
* Operations are methods, actions, or functions that the object can perform to modify itself or perform for some external effect.

Usefulness of Objects:

* Code organization and maintainability.
* Add clarity, reduce complexity.
* Simple rules allow complex interactions.
* Emphasizes data over procedure.
* Code modularity.
* Code reusability.
* Well-suited for databases.

Using an Object:

* We must initiate an object before using it.
* After initiating, we can access all its public properties and methods using " ->" after the instance.

This Operator:

* $this is a reference to the calling object.
* $this means a reference to current instance of this object.
* $this keyword is only valid inside the scope of a method.
* We cannot use $this keyword from outside the class.

Class Constants:

* Constant values on a per-class basis remaining the same and unchangeable.
* Use the const keyword like const ASC=1;
* Constant variables never use $ symbol to declare or use them.
* The value must be a constant expression, not (for example) a variable, a property, a result of a mathematical operation, or a function call.
* The variable's value cannot be a keyword (e.g. self, parent and static).
* Constants are called outside of the class by scope resolution operator (::) followed by class name like Animal::NAME not after the instance of the class.
* To access these constants from within the class, we reference them with the self-keyword.

Class Constants are Useful

* Properties that never change
* Error codes
* Data structure names
* And things that have no need to change at runtime.

Best Practice to Define Constants

* ALL\_CAPS\_SEPERATED\_BY\_UNDERSCORES
* Constants should only contain simple values such as strings, Booleans and integers. Arrays will not work.
* To define a constant followed with const like const NAME\_NO\_DOLLAR\_SIGN = 44;

Constructor:

* The \_\_construct() is a special method inside a class which is called constructor method. Whenever a new object is created from this class, this method will execute automatically.
* If a class contains two types of constructor like \_\_construct() and another is same as the class name like Factorial() PHP5 will give preference to the \_\_construct(() function and the other one will be ignored.
* If the subclass has no constructor in it, the constructor from superclass will be invoked.
* Parent’s constructors are not called implicitly if the child class defines a constructor.
* In order to run a parent constructor, a call to parent::\_\_construct() within the child constructor is required.
* If the child does not define a constructor then it may be inherited from the parent class just like a normal class method (if it was not declared as private).
* Private may only be accessed by the class that defines the member.

Destructor:

* Destroying an object.
* Automatically invoked at the end of the execution of script.

Property Visibility:

* Private: Properties or methods declared as private are not allowed to be called from outside the class.
* Public can be accessed everywhere. Any property or method which is not explicitly declared as private or protected is a public method.
* Protected can be accessed only within the class itself and by inherited and parent classes
* If declared using var, the property will be defined as public.
* In short

1. Public = Everywhere
2. Private = This class only
3. Protected = This class and subclass

Method Visibility:

* Class methods may be defined as public, private, or protected. Methods declared without any explicit visibility keyword are defined as public.

Encapsulation:

* Hide data from the outside world.
* Also known as data hiding.
* Data or properties only accessible by the related object method.

Polymorphism:

* Polymorphism is the process of creating several objects from specific base classes.
* Different classes can have different behaviors for the same operation.
* For example, moving a bicycle requires completely different actions from those required for moving a car, even though the concepts are similar.
* We can check whether a class is derived from a class by using instanceof operator like

if ($htmlemailer instanceof extendedEmailer )

echo "HTML Emailer is Derived from Emailer.<br/>";

Inheritance:

* Inheritance allows you to create a hierarchical relationship between classes using subclasses.
* A subclass inherits attributes and operations from its superclass. For example, car and bicycle have some things in common.
* Parent Class alias names are root, super, base class and Child class alias names are sub, derived class.
* Inheritance is appropriate in “is-a” relationship like “a car is a vehicle”.
* A class can inherit the methods and properties of another class by using the keyword extends.
* Not possible to extend multiple classes; a class can only inherit from one base class.
* When overriding methods, the parameter signature should remain the same or PHP will generate an E\_STRICT level error.
* Constructor allows overriding with different parameters.

Special notes about Inheritance:

* We will put our common method in parent and then inherit those method in our sub class.

Coupling

* Coupling is the behavior of how classes are dependent on each other. Loosely coupled architecture is much more reusable than tightly coupled objects.

Instance

Whenever you create an object by calling its constructor, it will be called an instance. To simplify this, whenever you write something like this $var = new Object(); you actually create an instance of object class.

Interface:

* Interface something like structure.
* An interface can extend an arbitrary number of other interfaces at a time.
* Interface is an empty class which contains only the declaration of methods.
* So any class which implements this interface must contain the declared functions in it.
* So, interface is nothing but a strict ruling, which helps to extend any class and strictly implement all methods defined in interface.
* A class can use any interface by using the implements keyword.
* In interface we can only declare methods, but we cannot write their body.
* That means the body of all methods must remain blank.
* Every method structure must exactly be the same as defined in the interface.

Abstraction:

* An abstract class is almost the same as interface, except that now the methods can contain body.
* An abstract class must also be "extended", not "implemented".
* We cannot declare an abstract class as final, because abstract class means it has to be extended and final class means it can't be extended.
* Classes defined as abstract may not be instantiated.
* Any class that contains at least one abstract method must also be abstract.
* Methods defined as abstract simply declare the method's signature - they cannot define the implementation.
* When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child.
* Additionally, these methods must be defined with the same (or a less restricted) visibility.
* For example, if the abstract method is defined as protected, the function implementation must be defined as either protected or public, but not private.
* Furthermore the signatures of the methods must match, i.e. the type hints and the number of required arguments must be the same.
* For example, if the child class defines an optional argument, where the abstract method's signature does not, there is no conflict in the signature.

Why static is needed?

* Single connections for databases or services
* Avoids need for global scope (before PHP 5.3)
* Single instances of a class
* No need to instantiate member class
* Properties are stored at class level not in object level
* Track number of instantiated objects

Access Static Property and Method

* Class::method()
* Class::$property

Static:

* A static variable exists only in a local function scope, but it does not lose its value when program execution leaves this scope.
* We can access a static method or property directly without creating any instance of that class.
* Static properties persist the last state of what it was assigned, which is very useful in some cases.
* You use static when you want to use a method / variable that is not tied to an instance.
* Static declarations are resolved in compile-time.
* Declaring class properties or methods as static makes them accessible without needing an instantiation of the class. A property declared as static cannot be accessed with an instantiated class object (though a static method can).
* The pseudo-variable $this is not available inside the method declared as static.
* Static properties cannot be accessed through the object using the arrow operator

->.

* Excessive static methods make no use at all. Unless you have any specific purpose, don't use static members.

PHP self-versus static:

* Self-references the current class.
* Whereas the static keyword allows the function to bind to the calling class at runtime. Ref: oop-concept/Others/StaticVersus Self
* static allows the function to bind to the calling class at runtime static will get function from the calling class if the method is exist in calling class or child class otherwise it will get method from the base class
* Anytime we use word self in static method it will refers actual objects resides.

Self-Versus $this:

* $this is used to reference the current object.
* $this always refers to current instance.
* Whereas self is used to access the current class itself.

Static Binding

* Static methods will be early binding.
* Avoid static methods

🡪$user = User::find\_by \_id(1);

* Valid approach

🡪$user = new User();

🡪$user = $User->find\_by\_id(1);

* Deduce the class name

🡪get\_class()

🡪In our perspective problem it will returns DatabaseObject.

* Late Static Bindings

🡪get\_called\_class()

🡪The function get\_called\_class() can be used to retrieve a string with the name of the called class and static:: introduces its scope.

🡪static::instead of self:: //binding are happening in the runtime not at the very beginning.

Static method differences

* Cannot use $this
* Fatal error
* Use self
* self::$property

Late static bindings:

* A call to a static function that is inherited will “bind” to the calling class at runtime

Scope Resolution Operator (::)

* It is a token that allows access to static, constant, and overridden properties or methods of a class.
* When referencing these items from outside the class definition, use the name of the class.
* It's possible to reference the class using a variable. The variable's value cannot be a keyword (e.g. self, parent and static).
* Scope resolution operator also used in the case of static operator and static method.
* In this case can’t use object for referencing the static method and static attributes.

Final keyword:

* Final keyword prevents child classes from overriding a method by prefixing the definition with final. If the class itself is being defined final then it cannot be extended.
* Preventing from Overriding - If you declare any method as a final method, it can't be overridden in any of its subclass.
* Preventing from Extending - We can declare a class as final, which will prevent anyone from extending it.

Classes/Object Functions:

* Is\_a() 🡪 Checks if the object is of this class or has this class as one of its parents. Example: is\_a($objectName, “ClassName”)

Accessor Methods:

* Accessor methods are simply methods that are solely devoted to get and set the value of any class properties.
* There are two types of accessor methods. One is called getter, whose purpose is returning value of any class property. The other is setter that sets a value into a class property.
* A setter method should start with set and the property name with the first character capitalized. A getter method should start with get followed by the variable name with the first letter capitalized.
* That means if we have a property named email, the getter method should be named as getEmail and the setter method should be named as setEmail.

Class Information Functions

Checking if a Class Already Exists

* class\_exists() function is to first check if a class is already available.

<?

include\_once("../ch2/class.emailer.php");

if( class\_exists("Emailer"))

{

$emailer = new Emailer("hasin@pageflakes.com");

}

else

{

die("A necessary class is not found");

}

?>

Finding Currently Loaded Classes:

* get\_declared\_classes() function -- which classes are loaded in the current scope.

Finding out if Methods and Properties Exists:

* method\_exists() and property\_exists() functions.

Checking the Type of Class:

* There is a function called is\_a() that you can use to check the type of class.

Finding out the Class Name:

* get\_class() function to identify the class name.

Autoload

* The \_\_autoload() function will internally search out the class and load its definition.
* Each class file should be named the same as the class definition itself. eg: a class definition file named fax would have the filename fax.class.php

Static

* The Static instance of a class only happens once, and its variables are available to any instance of the class. Instances have their own individual values which are not accessible to other instances, except where they are marked static.