Object

Objects have multiple attributes. A cup can be full or empty or somewhere in between, it could be glass or plastic or some other material or it could be large or small or somewhere in between. Objects also have behavior. Consider a bird can fly, but an ostrich can't. Behavior can also be specific to a type of object. These three things: identity, attributes and behavior are the aspects that describe an object in an Object-Oriented programming. Objects have unique identity; they have their own attributes, information describing their current state; and have their own behavior, things they can do.

Class

The Object-Oriented design is not about objects, it's about classes, since we use classes to create objects. A class is a blueprint, a definition. The class comes first, since they create objects. Classes have attributes sometimes referred to as properties and behavior sometimes referred to as operations. When behavior is written in code, we refer to this as methods. They can create instances of a class. Each object is an instance of a particular class or method. The process of creating these objects is referred to as instantiation.

Abstraction

The other four fundamental aspects of Object-Orientation have cryptic terms: abstraction, polymorphism, inheritance and encapsulation. Think of them as APIE. Abstraction supports two other fundamental aspects of Object-Orientation, such as, inheritance and polymorphism.

Encapsulation

Consider food containers containing other things, keeping the contents together and protected. Encapsulation combines attributes, behaviors and objects together in a common container. Encapsulation also restricts access of the inner workings of a class, method or any other object. This is referred to as information hiding or data hiding. It's not about being secretive, but reducing dependencies between different parts of an application. A change in one part of the code won't cascade, requiring multiple code changes. A good rule of thumb is to encapsulate as much as possible except where it's absolutely necessary to expose.

Inheritance

Child classes or methods inherit from a parent class or method, so our new child classes or methods have the same attributes and behaviors of the parent class or method, This reduces the need to write redundant code. Only minor changes are needed in the child classes or methods to accommodate the minor differences. This not only saves time and effort, but also allows the reuse code. When you change a parent class or method, you change all of the child classes or methods that inherit from the parent. Inheritance supports the last of the fundamental aspects, polymorphism.

Polymorphism

Polymorphisms create different results from the same function where classes or methods are similar. It's the most complex of the fundamental aspects to understand, but also very powerful. It provides the correct behavior even when working with different variable types and conditions. Consider the plus sign(+) and the following simple function:

```
<?php
        function combine($a, $b) {
                return $a + $b;
        }
?>
```

<?php echo combine(1, 2); ?> returns the value of 3.
<?php echo combine(“b”, “c”); ?> returns the value of bc.

Polymorphisms are flexible, and return the correct result for different variables and conditions, and once reduces the need to write redundant code. It’s good practice to write a master class and have all other classes inherit or extend from it