# fundamentos de POO. Polimorfismo.

Como parte del repaso de programación orientado objetos, veremos el tema de polimorfismo y un ejemplo de aplicación, al finalizar el repaso deberán realizar un ejemplo.

## Marco Teórico.

### Introducción.

El polimorfismo, junto con la encapsulación y la herencia, forman parte de los pilares básicos de la programación orientada a objetos.

El polimorfismo, como su nombre indica, sugiere múltiples formas. En programación cuando hablamos de polimorfismo nos referimos a la capacidad de acceder a múltiples funciones a través del mismo interfaz. Es decir que un mismo identificador, o función puede tener diferentes comportamientos en función del contexto en el que sea ejecutado.

El polimorfismo es un concepto difícil de entender en un primer momento y en mi opinión PHP no nos ayuda demasiado en este aspecto, al no ser este un lenguaje de programación fuertemente tipado.

Su implementación varía en función del lenguaje de programación. En algunos casos para establecer una relación polifórmica es necesario que cada uno de los objetos implicados compartan una misma raíz, siendo entonces necesario establecer una jerarquía de clases. Este es el caso de los lenguajes de programación fuertemente tipados como Java.

Para el caso que nos ocupa vamos a estudiar el polimorfismo en PHP estableciendo una jerarquía de clases, ya que nos resultara más fácil portar el mismo ejemplo a otros lenguajes.

### ¿Cómo se implementa el polimorfismo?

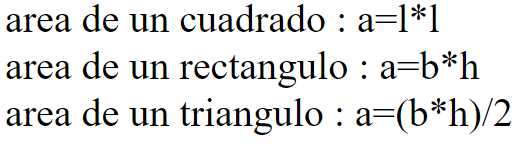
A continuación, veremos un ejemplo de cómo implementar una llamada polimórfica al método de un objeto.

Para ello vamos a crear una sencilla jerarquía de clases donde tendremos una clase base llamada "classPoligono" y sus respectivas clases extendidas: "classTriangulo" , "classCuadrado" , "classRectangulo". Cada una de estas clases tendrá un método en común que se llamará "calculo" y cuya función será la de mostrar la formula matemática para el cálculo del área de la figura geométrica en cuestión.

Una vez que hemos definido nuestras clases crearemos la función que se encargara de hacer la llamada polimórfica al método "calculo" cuya ejecución variara dependiendo del objeto que lo implementa.

1. **<?php**
2. */\**
3. *Empezaremos definiendo la jerarquia de clases*
4. *\*/*
5. **class**classPoligono
6. {
7. **function**calculo()
8. {
9. **echo**'El área depende del tipo de polígono';
10. }
11. }
12. **class**classCuadrado **extends**classPoligono
13. {
14. **function**calculo()
15. {
16. **echo**'área de un cuadrado : a=l\*l<br>';
17. }
18. }
19. **class**classRectangulo **extends**classPoligono
20. {
21. **function**calculo()
22. {
23. **echo**'área de un rectángulo : a=b\*h<br>';
24. }
25. }
26. **class**classTriangulo **extends**classPoligono
27. {
28. **function**calculo()
29. {
30. **echo**'área de un triángulo : a=(b\*h)/2<br>';
31. }
32. }
33. */\* fin definicion de la jerarquia de clases \*/*
34. */\**
35. *definicion de la funcion encargada de realizar las llamada*
36. *polimorfica al metodo "calculo"*
37. *A destacar que en la definicion de la funcion definimos el tipo*
38. *parametro que pasamos por referencia, esto no es obligatorio*
39. *en PHP, pero nos ayuda a entender el concepto y asi poder aplicarlo*
40. *en otros lenguajes mas estrictos.*
41. *\*/*
42. **function**area(classPoligono $obj)
43. {
44. $obj->calculo();
45. }
46. */\**
47. *Creamos los objetos necesarios*
48. *\*/*
49. $cuadrado = **new**classCuadrado;
50. $rectangulo = **new**classRectangulo;
51. $triangulo = **new**classTriangulo;
52. */\**
53. *Ejecutamos la funcion encargada*
54. *de realizar la llamada polimorfica*
55. *\*/*
56. area($cuadrado);
57. area($rectangulo);
58. area($triangulo);
59. **?>**

### Resultado



## conclusiones

Al ejecutar el ejemplo anterior vemos que la función "area" nos muestra la formula correcta en cada una de sus ejecuciones para cada tipo de figura geométrica, pese a que en su definición inicial hayamos especificado que el objeto es del tipo "classPoligono", haciendo referencia a la clase base de cada objeto. Esto sería necesario en algunos lenguajes en los cuales nos hace falta un nexo común a cada uno de los objetos, y la única forma es de utilizar la clase base común a cada uno de ellos.

## actividad

1. Realizar la función perímetro implementando herencia en el código anterior. (**para hoy antes de que acabe la clase**).  
   - El ejercicio se debe se debe enviar en un archivo zip comprimido con el nombre de DAII-Ejercicio03-NombreApellido.zip con el mismo asunto del correo.  
   - Debe contener todos los documentos necesarios para su ejecución.
2. Agregar valores predeterminados a las clases de las figuras geométricas e imprimir el área y perímetro de cada una con él la formula y el resultado numérico de la misma (**Para hoy antes del medio día**).

* Agregar todos los archivos necesarios para la ejecución en un zip con el siguiente nombre DAII-Ejercicio03B-NombreApellido.zip, y el asunto del correo debe ser el mismo que el nombre del archivo.

1. EXTRA: En la última versión si los valores se pueden leer a través del método GET.

## Rubrica.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Puntos | |  |
| Aspecto | 10 | 0 | Puntos |
| * 1. Adjunto los archivos necesarios para la ejecución del ejercicio DAII-Ejercicio03-NombreApellido.zip | Todos los archivos | Faltan archivos |  |
| * 1. El archivo realiza satisfactoriamente la salida esperada. | Completo y sin error. | Incompleto o sin error |  |
| * 1. Adjunto los archivos necesarios para la ejecución del ejercicio DAII-Ejercicio03-NombreApellido.zip | Todos los archivos | Faltan archivos |  |
| * 1. El archivo realiza satisfactoriamente la salida esperada. | Completo y sin error. | Incompleto o sin error |  |
| * 1. EXTRA. | Completo y sin error. | Incompleto o sin error |  |