# POLIMORFISMO

El **polimorfismo** es uno de los pilares fundamentales de la programación orientada a objetos (POO).  
La palabra proviene del griego y significa "*muchas formas*". En POO, el polimorfismo permite que **una misma operación se comporte de diferentes maneras según el objeto que la invoque**.

En C#, esto se logra generalmente a través de:

* **Herencia** y **métodos virtuales/override** (polimorfismo en tiempo de ejecución).
* **Sobrecarga de métodos** (polimorfismo en tiempo de compilación).

Con el polimorfismo, podés escribir código más flexible, reutilizable y fácil de extender.

Podemos decir que el polimorfismo en programación es la funcionalidad que permite al código antiguo invocar un nuevo código. Este es probablemente el mejor beneficio de la programación orientada a objetos porque permite mejorar y extender el código sin tener que adaptar el código ya existente.

## Polimorfismo en c# y palabras reservadas

En el polimorfismo existen 3 palabras reservadas:

**Base**, **Virtual** y **override**. La palabra reservada **BASE** se utiliza para hacer referencia a la clase padre y poder acceder a sus métodos y atributos públicos o protegidos, por ejemplo:

// Clase base

   class operacion{  
        **public** void **mostrar**(){  
            Console.**WriteLine**("El resultado de la operación es:");  
        }  
    }  
    // Clase derivada 1  
    class suma:operacion{  
        **public**  void **Sumar**(){  
            **base**.**mostrar**();  
            Console.**WriteLine**(2+3);  
        }  
    }

class Program  
    {  
        **public** static void **Main**(string[] args)  
        {  
            suma objsuma =**new** suma();  
            objsuma.**Sumar**();  
        }  
    }

Mientras que la palabra reservada **virtual** se utiliza para indicar que el método de una clase puede ser sobrescrito por otro método del mismo nombre en una clase hija.

Y la palabra **override** es para indicar que un método sobrescribe al método del mismo nombre que exista en la clase padre. Por ejemplo

// Clase base  
**public** class Animal  
{  
    **public** virtual void **HacerSonido**()  
    {  
        Console.**WriteLine**("El animal hace un sonido");  
    }  
}  
  
// Clase derivada 1  
**public** class Perro : Animal  
{  
    **public** override void **HacerSonido**()  
    {  
        Console.**WriteLine**("El perro dice: ¡Guau!");  
    }  
}  
  
// Clase derivada 2  
**public** class Gato : Animal  
{  
    **public** override void **HacerSonido**()  
    {  
        Console.**WriteLine**("El gato dice: ¡Miau!");  
    }  
}  
  
// Programa principal  
class Program  
{  
    static void **Main**()  
    {  
        // Usamos polimorfismo: una variable de tipo base puede referenciar objetos derivados  
        Animal miAnimal1 = **new** Perro();  
        Animal miAnimal2 = **new** Gato();  
  
        miAnimal1.**HacerSonido**();  // Salida: El perro dice: ¡Guau!  
        miAnimal2.**HacerSonido**();  // Salida: El gato dice: ¡Miau!  
    }  
}

En c# las palabras **override** y **virtual** son opcionales puesto que c# es un lenguaje orientado a objetos, entiende que cuando creamos un método con el mismo nombre en una clase hija y una clase padre estamos indicando que lo estamos sobrescribiendo.

**📝 Ejercitación**

Una empresa que se dedica a la producción de muebles fabrica los siguientes productos:

Mesa: color, m2 de madera, tipo (circular o rectangular).

Banco: color, m2 de madera, tipo (interior o exterior), con o sin tapizado.

Placar: color, m2 de madera, cantidad de cajones.

La aplicación debe permitir calcular la cantidad de madera y pintura como el costo total para cada producto.

La madera cuesta 125$ el m2 y la pintura $78 el litro. Tener en cuenta que por cada m2 de madera normalmente se usa 600ml de pintura

Banco: un banco ocupa 0.7m2 de madera, si es con tapizado, sino ocupa unos 0.9 m2. Dependiendo del tipo de banco tener en cuenta que si es exterior va a ocupar 1 mano extra de pintura lo que equivale a 50% más de pintura.

Mesa: si la mesa es rectangular la aplicación debe solicitar largo y alto, caso contrario deberá solicitar el radio

Placar: tener en cuenta que el placar tiene 4 caras rectangulares las de arriba y abajo son del mismo tamaño al igual que las de la izquierda con las de la derecha, se debe solicitar el largo y alto de cada cara. tener en cuenta que cada cajón ocupa 0,7 m2 extra de madera.