# Polimorfismo

* Es el concepto que se refiere a la habilidad de poder realizar una operación con objetos de distintos tipos.
* El término proviene de las raíces griegas Poli + Morfo. Significa "cualidad de tener muchas formas".

La herencia nos permite heredar campos y métodos de otra clase. El polimorfismo usa esos métodos para realizar diferentes tareas. Esto nos permite realizar una misma acción de diferentes maneras.

## Beneficios

* Permite crear sistemas escalables
* Objetos de distintos tipos se comportan como uno solo.
* Reúso de código.

Por ejemplo, piensa en una clase base llamada Animal que tiene un método llamado sonidoAnimal(). Las clases derivadas de animales podrían ser gatos, perros, osos, pájaros, y también tienen su propia implementación de un sonido animal (el perro ladra y el gato maúlla, etc.):

class Animal // Clase base (padre)

{

public void sonidoAnimal()

{

Console.WriteLine("El animal hace un sonido");

}

}

class Gato: Animal // Clase derivada (hijo)

{

public void sonidoAnimal()

{

Console.WriteLine("El gato dice: miau miau");

}

}

class Perro : Animal // Clase derivada (hijo)

{

public void sonidoAnimal()

{

Console.WriteLine("El perro dice: guau guau");

}

}

## Interfaces

Las interfaces son la forma en que C# implementa el polimorfismo.

* Definen métodos y propiedades completamente abstractos (con cuerpos vacíos), pero no campos/variables.
* Se comportan como un contrato.
  + Una clase que implementa una interfaz debe sobrescribir todos sus métodos y propiedades.
* Una clase puede implementar múltiples interfaces, para implementar múltiples interfaces, sepárelas con una coma.
* De forma predeterminada, los miembros de una interfaz son abstract y public.
* Una interfaz no puede contener un constructor (ya que no se puede usar para crear objetos)

Por convención en C# todas las interfaces deben comenzar con la I mayúscula de Interface.

**Ejemplo**

interface IFigura

{

double CalcularArea();

}

class Circulo : IFigura

{

public double Radio { get; set; }

public double CalcularArea()

{

return Math.PI \* Radio \* Radio;

}

}

No es necesario poner override a la hora de implementar o sobreescribir el método.

## Clases abstractas

* Una clase abstracta tiene la característica de que no puede ser instanciada, es decir, no podemos crear un objeto a partir de esa clase.
* El objetivo es crear subclases que completen a esa clase abstracta y le den la funcionalidad específica que necesita.
* Una clase abstracta es incompleta y va a estar completa hasta que nosotros definamos alguna subclase y esa subclase sobrescriba métodos o propiedades que la clase abstracta dejó definida.

La palabra reservada abstract se utiliza para clases y métodos:

**Clase abstracta:** es una clase restringida que no se puede usar para crear objetos (para acceder a ella, se debe heredar de otra clase).

**Método abstracto:** solo se puede usar en una clase abstracta y no tiene cuerpo. El cuerpo lo proporciona la clase derivada (heredada de).

Una clase abstracta puede tener métodos abstractos y regulares:

abstract class Animal

{

public abstract void sonidoAnimal();

public void dormir()

{

Console.WriteLine("Zzz");

}

}

## Clase virtual

La palabra reservada virtual se usa para modificar una declaración de método, propiedad, indizador o evento y permitir que se invalide en una clase derivada. Por ejemplo, cualquier clase que herede este método puede reemplazarlo:

# Bibliografía

<https://www.w3schools.com/cs/cs_polymorphism.php>

<https://www.udemy.com/course/c-sharp-para-principiantes/learn/lecture/4618968>