

# OOP - C#

Jorge Mario Londoño Peláez & Varias AI

February 3, 2025

## 1 Repaso Conceptos OOP - Version C#

### 1.1 Qué es la programación orientada a objetos

La Programación Orientada a Objetos (OOP) es un paradigma de programación que se basa en el concepto de "objetos", los cuales son entidades que combinan datos (atributos) y comportamiento (métodos). A diferencia de la programación procedimental, que se centra en funciones o procedimientos, la OOP organiza el código en torno a objetos que interactúan entre sí. En comparación con la programación funcional, que se enfoca en la evaluación de funciones y la inmutabilidad, la OOP permite modelar el mundo real de manera más intuitiva mediante la creación de objetos que representan entidades del problema.

### 1.2 Clases y Objetos

En C#, una **clase** es una plantilla o un plano para crear objetos. Define las características (atributos) y los comportamientos (métodos) que tendrán los objetos de esa clase. Un **objeto** es una instancia específica de una clase.

La palabra clave **this** se utiliza dentro de un método de instancia para referirse al objeto actual. Permite acceder a los miembros de la clase desde dentro de la misma.

#### 1.2.1 Ejemplo: Definición e instancias de una Clase

```
1 public class Coche {
2     public string marca;
3     public string modelo;
4
5     public Coche(string marca, string modelo) {
6         this.marca = marca;
7         this.modelo = modelo;
8     }
9
10    public void Acelerar() {
11        Console.WriteLine("El coche " + marca + " " + modelo + " esta acelerando.");
12    }
13 }
14
15 public class Ejemplo {
16     public static void Main(string[] args) {
17         Coche coche1 = new Coche("Toyota", "Corolla");
18         Coche coche2 = new Coche("Honda", "Civic");
19     }
```

```

20     coche1.Acelerar(); // Output: El coche Toyota Corolla esta acelerando.
21     coche2.Acelerar(); // Output: El coche Honda Civic esta acelerando.
22 }
23 }

```

Listing 1: Ejemplo de clase y objetos en C#

### 1.2.2 Variables y métodos estáticos

Las variables y métodos estáticos pertenecen a la clase en sí, en lugar de a una instancia específica de la clase. Se acceden utilizando el nombre de la clase en lugar del nombre de un objeto.

```

1 public class Contador {
2     public static int conteo = 0;
3
4     public static void Incrementar() {
5         conteo++;
6     }
7 }
8
9 public class Ejemplo {
10     public static void Main(string[] args) {
11         Contador.Incrementar();
12         Console.WriteLine(Contador.conteo); // Output: 1
13         Contador.Incrementar();
14         Console.WriteLine(Contador.conteo); // Output: 2
15     }
16 }

```

Listing 2: Ejemplo de variables y métodos estáticos en C#

## 1.3 Encapsulamiento

El encapsulamiento es el principio de ocultar los detalles internos de un objeto y exponer solo la información necesaria. En C#, se logra mediante el uso de modificadores de acceso como `public`, `private`, `protected` e `internal`. Esto permite controlar cómo se accede y se modifica el estado de un objeto, protegiéndolo de manipulaciones no deseadas.

## 1.4 Herencia

La herencia es un mecanismo que permite crear nuevas clases (subclases o clases derivadas) basadas en clases existentes (superclases o clases base). Las subclases heredan los atributos y métodos de sus superclases, lo que fomenta la reutilización de código y la creación de jerarquías de clases.

### 1.4.1 Ejemplo de Herencia en C#

```

1 public class Animal {
2     public string nombre;
3
4     public Animal(string nombre) {
5         this.nombre = nombre;
6     }
7
8     public virtual void HacerSonido() {
9         Console.WriteLine("Sonido generico de animal");

```

```

10     }
11 }
12
13 public class Perro : Animal {
14     public Perro(string nombre) : base(nombre) { }
15
16     public override void HacerSonido() {
17         Console.WriteLine("Guau!");
18     }
19 }
20
21 public class Gato : Animal {
22     public Gato(string nombre) : base(nombre) { }
23
24     public override void HacerSonido() {
25         Console.WriteLine("Miau!");
26     }
27 }
28
29 public class Ejemplo {
30     public static void Main(string[] args) {
31         Animal animal = new Animal("Animal");
32         Perro perro = new Perro("Firulais");
33         Gato gato = new Gato("Michi");
34
35         animal.HacerSonido(); // Output: Sonido generico de animal
36         perro.HacerSonido();  // Output: Guau!
37         gato.HacerSonido();    // Output: Miau!
38     }
39 }

```

Listing 3: Ejemplo de herencia en C#

## 1.5 Clases Abstractas

Una clase abstracta es una clase que no se puede instanciar directamente. Se utiliza como una plantilla para otras clases, y puede contener métodos abstractos (métodos sin implementación). Las clases que heredan de una clase abstracta deben implementar todos sus métodos abstractos.

### 1.5.1 Ejemplo: Definición de superclase abstracta y subclase concreta

```

1 public abstract class Figura {
2     public abstract double CalcularArea();
3 }
4
5 public class Circulo : Figura {
6     public double radio;
7
8     public Circulo(double radio) {
9         this.radio = radio;
10    }
11
12    public override double CalcularArea() {
13        return Math.PI * radio * radio;
14    }
15 }
16

```

```

17 public class Ejemplo {
18     public static void Main(string[] args) {
19         Circulo circulo = new Circulo(5);
20         Console.WriteLine("Area del circulo: " + circulo.CalcularArea()); //
21         Output: Area del circulo: 78.53981633974483
22     }
23 }

```

Listing 4: Ejemplo de clase abstracta en C#

## 1.6 Polimorfismo

El polimorfismo es la capacidad de un objeto de tomar muchas formas. En C#, se logra mediante la herencia y la implementación de interfaces. Permite tratar objetos de diferentes clases de manera uniforme, siempre y cuando compartan una interfaz común.

### 1.6.1 Ejemplo de Polimorfismo en C#

```

1 public interface IFigura {
2     double CalcularArea();
3 }
4
5 public class Rectangulo : IFigura {
6     public double ancho;
7     public double alto;
8
9     public Rectangulo(double ancho, double alto) {
10         this.ancho = ancho;
11         this.alto = alto;
12     }
13
14     public double CalcularArea() {
15         return ancho * alto;
16     }
17 }
18
19 public class Triangulo : IFigura {
20     public double baseTriangulo;
21     public double altura;
22
23     public Triangulo(double baseTriangulo, double altura) {
24         this.baseTriangulo = baseTriangulo;
25         this.altura = altura;
26     }
27
28     public double CalcularArea() {
29         return 0.5 * baseTriangulo * altura;
30     }
31 }
32
33 public class Ejemplo {
34     public static void Main(string[] args) {
35         IFigura[] figuras = new IFigura[2];
36         figuras[0] = new Rectangulo(5, 10);
37         figuras[1] = new Triangulo(4, 6);
38
39         foreach (IFigura figura in figuras) {

```

```

40         Console.WriteLine("Area: " + figura.CalcularArea());
41     }
42     // Output: Area: 50
43     // Output: Area: 12
44 }
45 }

```

Listing 5: Ejemplo de polimorfismo en C#

## 1.7 Interfaces

Una interfaz es un contrato que define un conjunto de métodos que una clase debe implementar. A diferencia de las clases abstractas, las interfaces no pueden contener ninguna implementación de métodos. Una clase puede implementar múltiples interfaces.

### 1.7.1 Ejemplo: Definición e implementación de una interfaz en C#

```

1 public interface ITransporte {
2     void Arrancar();
3     void Detener();
4 }
5
6 public class Coche : ITransporte {
7     public void Arrancar() {
8         Console.WriteLine("El coche ha arrancado.");
9     }
10
11     public void Detener() {
12         Console.WriteLine("El coche se ha detenido.");
13     }
14 }
15
16 public class Bicicleta : ITransporte {
17     public void Arrancar() {
18         Console.WriteLine("La bicicleta ha comenzado a rodar.");
19     }
20
21     public void Detener() {
22         Console.WriteLine("La bicicleta se ha detenido.");
23     }
24 }
25
26 public class Ejemplo {
27     public static void Main(string[] args) {
28         ITransporte coche = new Coche();
29         ITransporte bicicleta = new Bicicleta();
30
31         coche.Arrancar();    // Output: El coche ha arrancado.
32         bicicleta.Arrancar(); // Output: La bicicleta ha comenzado a rodar.
33     }
34 }

```

Listing 6: Ejemplo de interfaz en C#

## 1.8 Excepciones

Las excepciones son errores que ocurren durante la ejecución de un programa. C# proporciona mecanismos para manejar excepciones utilizando bloques `try-catch-finally`.

### 1.8.1 Atrapar excepciones

El bloque `try` contiene el código que puede generar una excepción. El bloque `catch` se utiliza para atrapar y manejar la excepción. El bloque `finally` se ejecuta siempre, independientemente de si se produjo una excepción o no.

```
1 public class Ejemplo {
2     public static void Main(string[] args) {
3         try {
4             int resultado = 10 / 0; // Esto generara una excepcion
              DivideByZeroException
5         } catch (DivideByZeroException ex) {
6             Console.WriteLine("Error: Division por cero. " + ex.Message);
7         } finally {
8             Console.WriteLine("Bloque finally ejecutado.");
9         }
10    }
11 }
```

Listing 7: Ejemplo de manejo de excepciones en C#

### 1.8.2 Crear excepciones

Se pueden crear excepciones personalizadas heredando de la clase `Exception` o de alguna de sus subclases.

```
1 public class MiExcepcion : Exception {
2     public MiExcepcion(string mensaje) : base(mensaje) { }
3 }
```

Listing 8: Ejemplo de creación de excepción personalizada en C#

### 1.8.3 Lanzar excepciones

Se utiliza la palabra clave `throw` para lanzar una excepción.

```
1 public class Ejemplo {
2     public static void Main(string[] args) {
3         try {
4             throw new MiExcepcion("Esta es mi excepcion personalizada.");
5         } catch (MiExcepcion ex) {
6             Console.WriteLine("Excepcion atrapada: " + ex.Message);
7         }
8     }
9 }
```

Listing 9: Ejemplo de lanzamiento de excepción en C#

## 1.9 Clases internas

Una clase interna (o clase anidada) es una clase que se define dentro de otra clase. Las clases internas pueden acceder a los miembros de la clase contenedora, incluso si son privados. Se utilizan para agrupar clases relacionadas y para implementar patrones de diseño específicos.

### 1.9.1 Ejemplo de una clase interna

```
1 public class ClaseExterna {
2     private int datoExterno = 10;
3
4     public class ClaseInterna {
5         public void MostrarDato() {
6             ClaseExterna claseExterna = new ClaseExterna();
7             Console.WriteLine("Dato externo: " + claseExterna.datoExterno);
8         }
9     }
10
11     public static void Main(string[] args) {
12         ClaseInterna claseInterna = new ClaseInterna();
13         claseInterna.MostrarDato(); // Output: Dato externo: 10
14     }
15 }
```

Listing 10: Ejemplo de clase interna en C#