Programación Orientada a Objetos en C#

Miguel Teheran

Prerrequisitos

Que necesitas para tomar este curso

Prerrequisitos

- Conocimientos de C# y .NET
- .NET 7 o superior
- Visual Studio Code o Visual Studio

¿Qué es la programación orientada a objetos (POO)?

Concepto

Paradigma de la programación

Formas o estilos en los que podemos programar y estructurar nuestro código.

Paradigmas importantes

- Programación funcional
- Programación estructurada
- Programación reactiva
- Programación orientada a aspectos
- Programación orientada a objetos

C# es multi-paradigma

Programación funcional

Programación orientada a objetos

Programación estructurada

Programación reflectiva

Programación imperativa

Programación genérica

Concepto de POO - OOP

- Paradigma de la programación.
- El objetivo es extraer elementos de la realidad a los algoritmos en nuestro código.
- Basado en clases y objetos.

Lenguajes que utilizan POO

- Java
- C++
- Ruby
- Python
- Visual Basic
- Perl

Creando tu primera clase y objeto

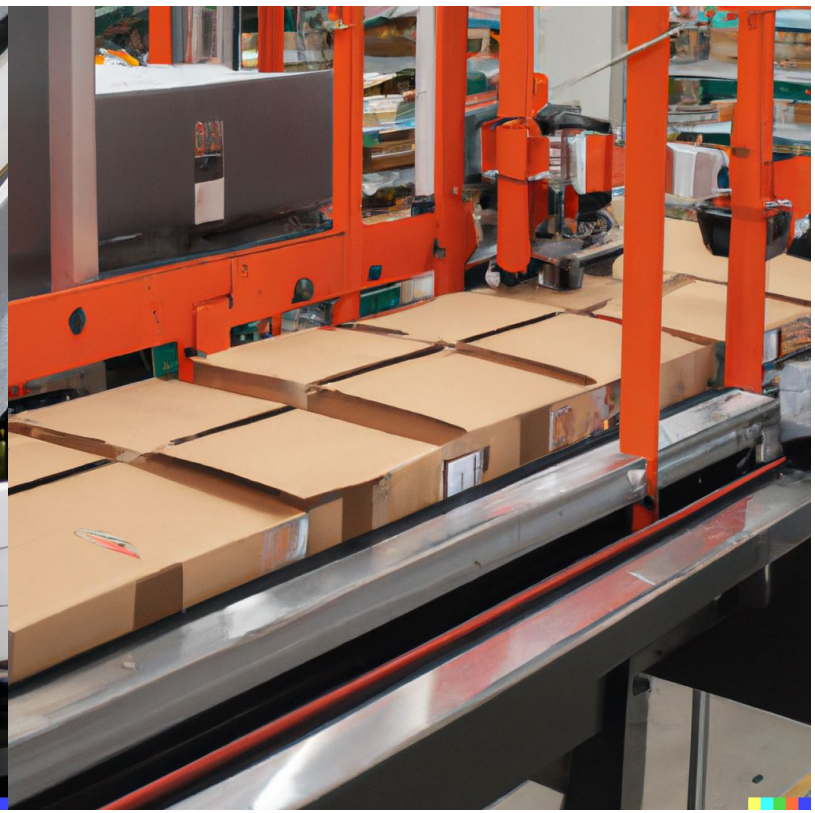
Concepto de clases y objetos

Concepto de clase

- Elemento principal en POO.
- Se basa en propiedades que representan las características del objeto y métodos que definen el comportamiento.
- Se pueden definir como la plantilla base para crear los objetos.

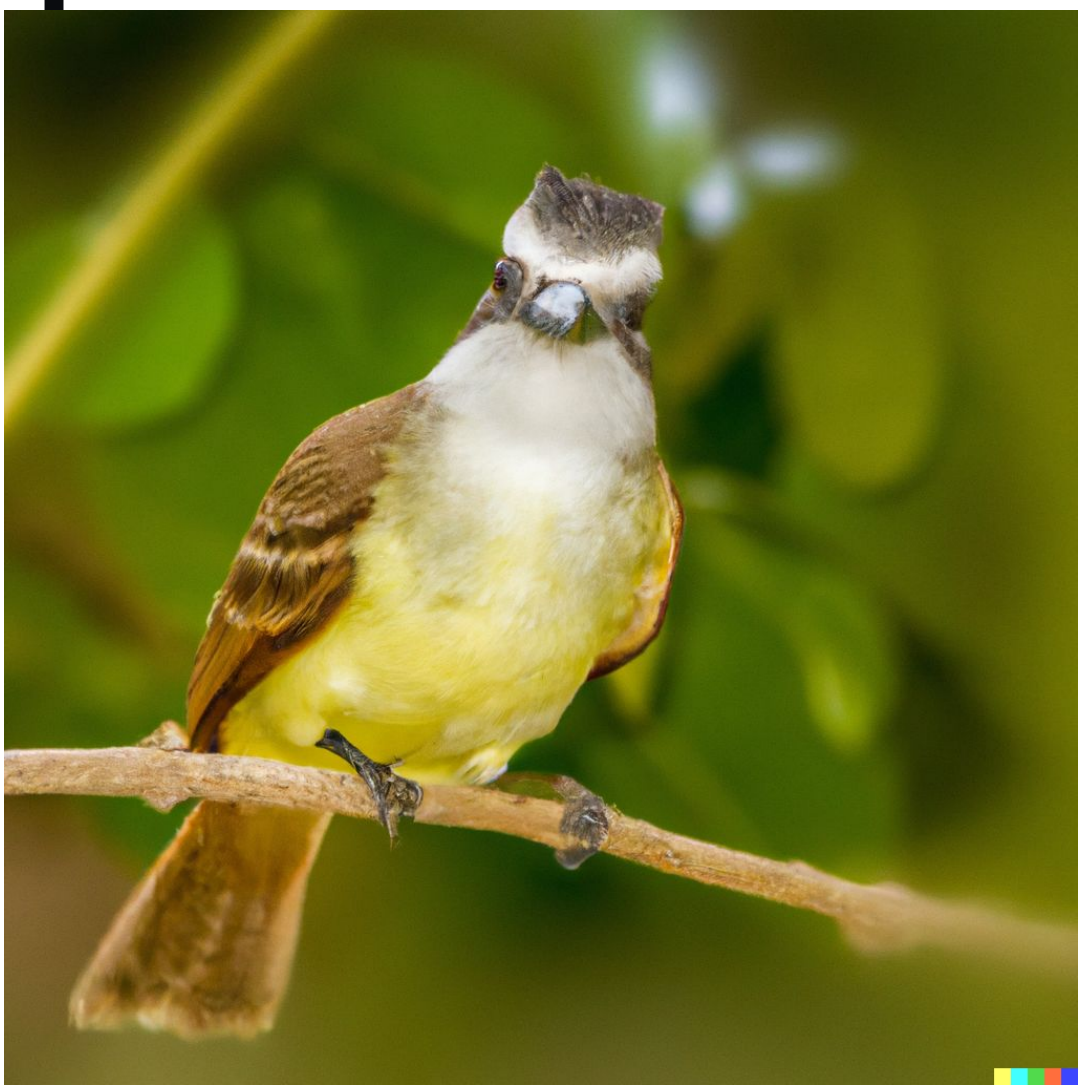
Los objetos son instancias de las clases

Un molde para generar piezas iguales



Los objetos se usan para guardar datos o ejecutar acciones que define la clase

Análisis de objetos - ejemplo de ave



Análisis de objetos - ejemplo de ave

Clase Ave

- Nombre
- Color
- Tamaño
- Familia
- Ubicación
- SexoBiológico
- EsVoladora

Análisis del proyecto propuesto usando POO

Concepto

Necesitamos crear una app para administrar superheroes

1. La app debe permitir crear diferentes superheroes con diferentes caracteristicas

2. La app debe permitir imprimir registro de las acciones que hacen los super heroes

3. La app debe poderse extender utilizando las caracteristicas de POO

Creando clases y propiedades

Constructor y datos iniciales en una clase

Constructor inicia los valores o especifica los datos por defecto de una clase

Ejemplo constructor

```
public class Persona {
   private string apellido;
   private string nombre;

   public Persona() {
     apellido="";
     nombre="";
}
```

Métodos dentro de una clase

Tipos registro y estructura

Tabla comparativa

Clase	**Estructura**	**Registro**
Referencia	Valor	Valor o referencia
Grandes	Pequeñas	Pequeñas
Valores y comportamientos	Enfocada a valores	Enfocada a valores inmutables

Modificadores de acceso

Quienes tienen accesos a las clases, propiedades y métodos

Lista de modificadores

- public
- protected
- internal
- private
- file

Lista de niveles de acceso

- **public:** el acceso no está restringido.
- **protected**: el acceso está limitado a la clase contenedora o a los tipos derivados de la clase contenedora.
- **internal**: el acceso está limitado al ensamblado actual.

Lista de niveles de acceso

- **protected internal**: El acceso está limitado al ensamblado actual o a los tipos derivados de la clase contenedora.
- **private**: el acceso está limitado al tipo contenedor.
- **private protected:** El acceso está limitado a la clase contenedora o a los tipos derivados de la clase contenedora que hay en el ensamblado actual.

Encapsulamiento

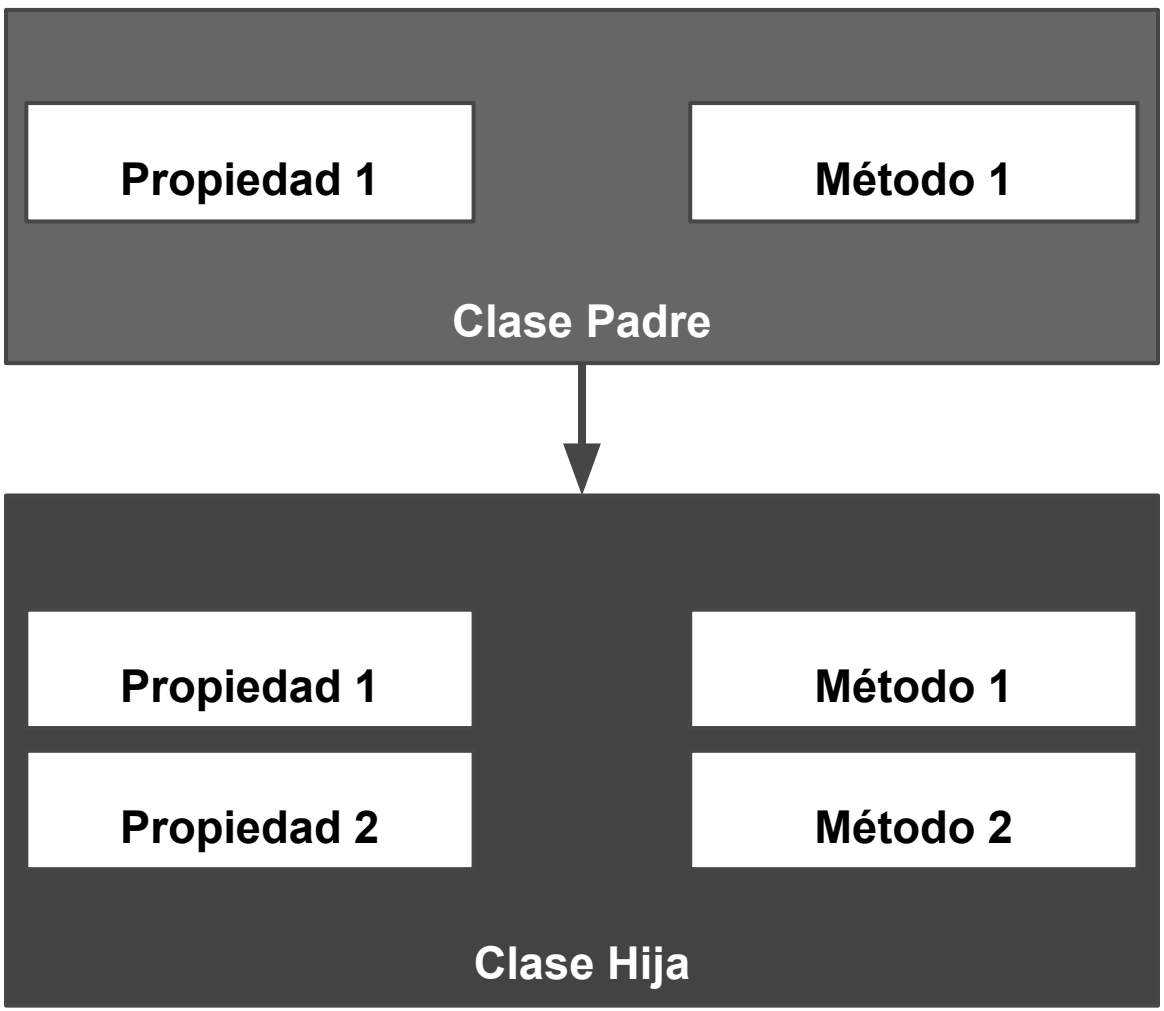
¿Que es herencia?

Herencia genetica

Herencia cultural

Herencia de patrimonio

Herencia en POO



Herencia en C#

```
public class ClaseHija : ClasePadre
```

En C# solo podemos heredar de una clase

Usando herencia en C#

Antihéroe

Abstracción

Polimorfismo

¿Qué es una interfaz?

Interfaz

- Un contrato para garantizar la estructura de una clase.
- Permite elementos abstractos e implementaciones por defecto.
- Ayudan a desacoplar el código.
- C# soporta implementación de múltiples interfaces.

S.O.L.I.D
Interface Segregation Principle

Características

- Normalmente se declaran con **I**Nombre
- Se utilizan en casi todos los patrones de diseño para .NET
- Podemos implementar **generics** para utilizar la misma interfaz en diferentes escenarios. Ejemplo: **IList<T>**

Interfaces vs Clases Abstractas

Comparativa

Clase abstracta	**Interfaz**
Solo permite una herencia	Permite múltiples implementaciones
Permite tener implementaciones	No es posible implementar (se puede desde C# 8.0)
Recomendada para reutilizar código y lógica	Recomendada para implementar patrones de diseño y la inyección de dependencias
Menos usada	Altamente usada

C# 8 - default interface methods

```
interface IA
{
    void M() { WriteLine("IA.M"); }
}
```

C# 8 - default interface methods

```
class C : IA { } // OK


IA i = new C();
i.M(); // prints "IA.M"
```