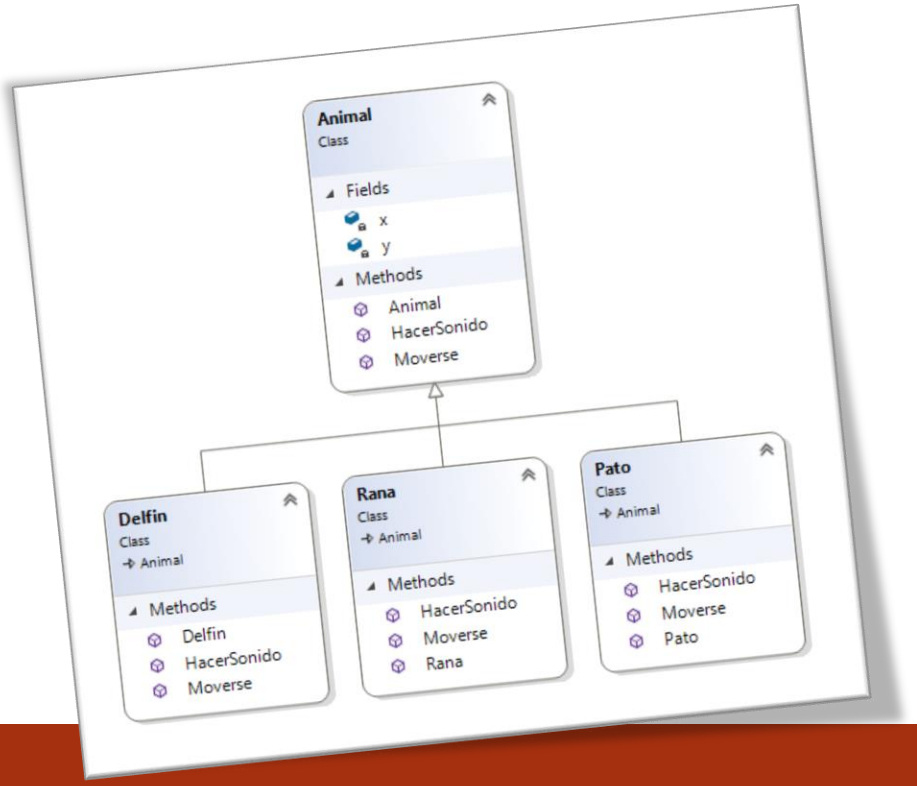
"La Ciencia sin Moral es Vana"

Universidad Católica de El Salvador
Facultad de Ingeniería y Arquitectura
Materia: Programación II
Docente: Master Giovanni Acosta

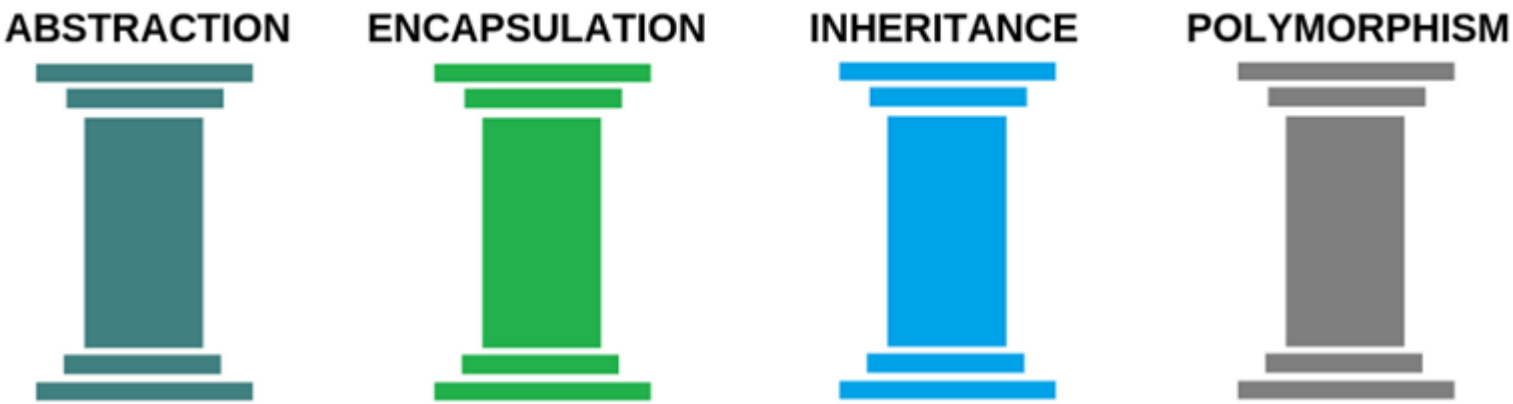
Tema 11: Polimorfismo en programación orientada a objetos



Objetivos:

- Conocer los beneficios del polimorfismo en la programación orientada a objetos
- Crear métodos virtuales
- Aplicar sobrecarga y sobreescritura de métodos
- Crear clases y métodos abstractos

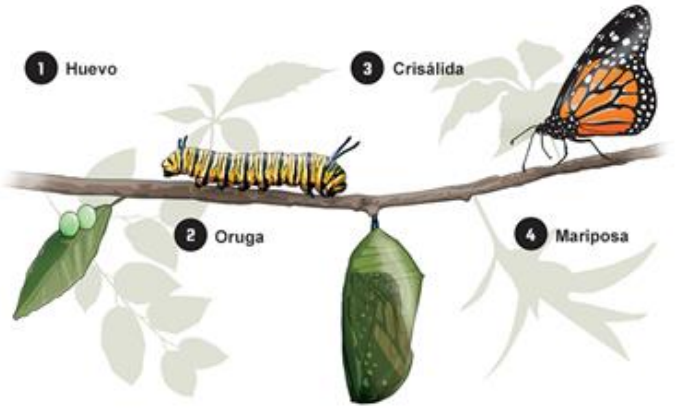
PILARES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS



¿QUÉ ES EL POLIMORFISMO?

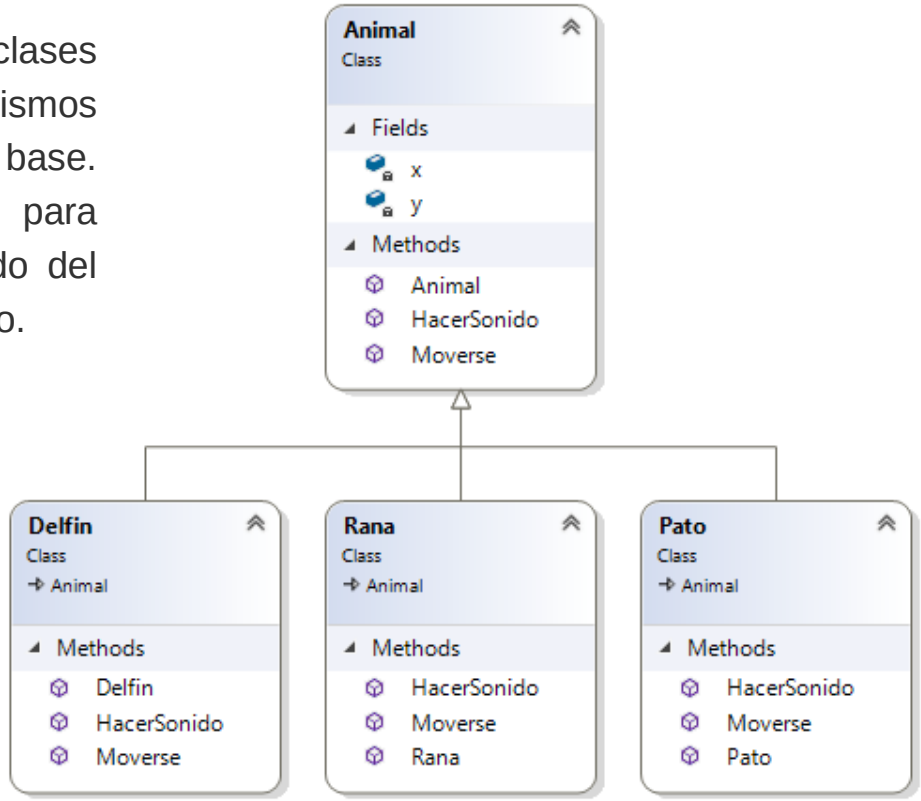
- La primera definición de polimorfismo en el diccionario de la real academia de la lengua española es **cualidad de lo que tiene o puede tener distintas formas**.

- Otro significado de polimorfismo en el diccionario es propiedad de las especies de seres vivos cuyos individuos pueden **presentar diferentes formas o aspectos**, bien por diferenciarse en castas, como las termitas, bien por tratarse de distintas etapas del ciclo vital, como la oruga y la mariposa.



¿QUÉ ES EL POLIMORFISMO EN POO?

- El término **polimorfismo** se refiere a la capacidad de un objeto para tomar diferentes formas (tipo o subtipo de objeto).
- El **polimorfismo** permite que las clases derivadas tengan métodos con los mismos nombres que los métodos en su clase base. Le da la capacidad a un programa para llamar al método correcto, dependiendo del tipo de objeto que se utiliza para llamarlo.

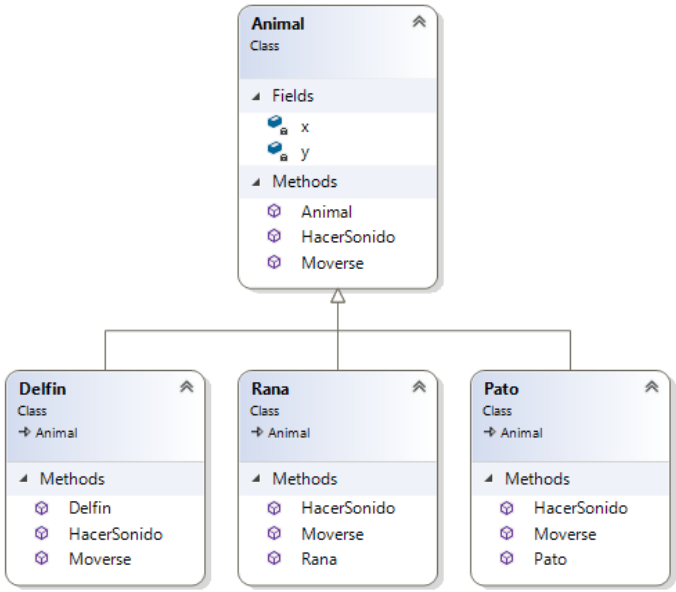


- Ejemplo: simulador de animales.

CARACTERÍSTICAS DEL POLIMORFISMO

1. La capacidad de definir un método en una clase base y luego definir un método con el **mismo nombre** en una clase derivada. Cuando un método de la clase derivada tiene el mismo nombre que un método de clase base, a menudo se dice que *el método de la clase derivada sobrescribe el método de clase base*.
2. La capacidad de llamar a la versión correcta de un **método sobrescrito**, dependiendo del tipo de objeto que se utiliza para invocarlo. Si se utiliza un objeto de la clase derivada para llamar a un método sobrescrito, entonces la versión de la clase derivada del método es la que se ejecuta. Si se utiliza un objeto de clase base para llamar a un método sobrescrito, entonces la versión de la clase base del método es la que se ejecuta.

```
Animal animal = new Animal(0, 0);
animal.Moverse();
animal.HacerSonido();

animal = new Delfin(5, 20);
animal.Moverse();
animal.HacerSonido();

animal = new Rana(25, 50);
animal.Moverse();
animal.HacerSonido();

animal = new Pato(75, 80);
animal.Moverse();
animal.HacerSonido();
```



- Ejemplo: simulador de animales.

¿QUÉ ES LA FIRMA (SIGNATURA) DE UN MÉTODO?

- La firma del método esta compuesta por las siguientes partes:
  1. El nivel de acceso: public o private y modificadores opcionales como abstract o sealed
  2. El tipo de retorno
  3. El nombre del método, y
  4. La lista de parámetros

- Ejemplo:

```
public void Abono(decimal cantidad)
{
    if (cantidad > 0 )
    {
        saldo += cantidad;
    }
}
```

<---------- Firma del método

<---------- Cuerpo o implementación del método

- Analizar la siguiente firma del método:

```
public decimal Pago(int cantidad, decimal precio)
{
    return cantidad * precio * 1.13m;
}
```

MÉTODOS VIRTUALES

- Son métodos en la clase base pensados para ser sobrescritos por las clases derivadas.
- Para declararlos, se utiliza la palabra reservada "**virtual**"; para sobrescribirlos, en la clase derivada se utiliza la palabra reservada "**override**".
- Un método virtual puede ser sobrescrito en la clase derivada, o utilizarse tal como está implementado en la clase base.
- Solo se puede utilizar "override" si el método en la clase base está marcado como "**virtual**", "**abstract**" u "**override**".

- En la clase base:

```
public virtual tipoDato NombreMetodo(listaParametros)
{
   //implementación del método
}
```

Se debe mantener la misma firma del método, en la clase base y en la clase derivada

- En la clase derivada:

```
public override tipoDato NombreMetodo(listaParametros)
{
   //sobreescritura del método
}
```

Con la palabra clave **base**.NombreMetodo, se puede acceder al método ocultado en la clase base

SOBREESCRITURA DE PROPIEDADES

- Las **propiedades** de una clase base pueden sobrescribirse de la misma manera en que se sobrescriben los métodos.
- En la clase base, se escribe la palabra clave "**virtual**" en la declaración de la propiedad, y en la clase derivada se escribe la palabra clave "**override**" para sobrescribirla.

- En la clase base:

```
public virtual double Peso
{
    get { return peso; }
    set { peso = value; }
}
```

- En la clase derivada:

```
public override double Peso
{
    get { return peso * 0.453592; }
    set { peso = value; }
}
```

SOBRECARGA (OVERLOAD) Y SOBRESCRITURA (OVERRIDING)

- Existe **sobrecarga** (overload) cuando una clase contiene dos o más métodos con el mismo nombre pero con distinta firma.

- Ejemplo:

```
public int Sumar(int n1, int n2) { return n1 + n2; }
public int Sumar(int n1, int n2, int n3) { return n1 + n2 + n3; }
public decimal Sumar(decimal n1, decimal n2) { return n1 + n2; }
```

- Existe **sobrescritura** (overriding) de propiedades y métodos cuando han sido heredados de la clase base y son sobrescritos (override) en la clase derivada.

- En la clase base:

```
public virtual int Pos(int x, int y)
{
    return x + y;
}
```

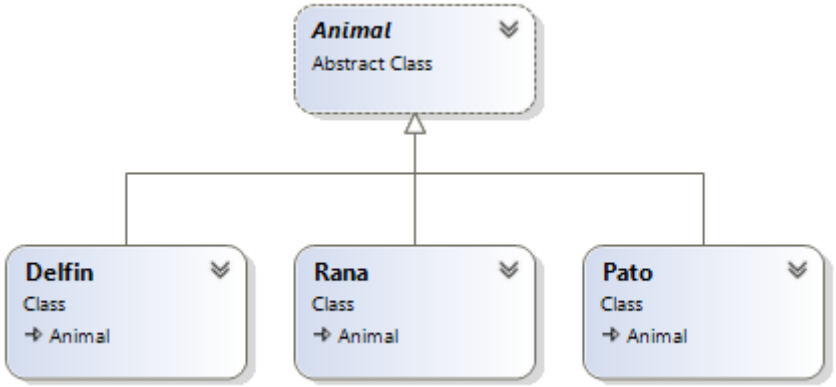
En la clase derivada:

```
public override int Pos(int x, int y)
{
    return (x + y)/3;
}
```

CLASES Y MÉTODOS ABSTRACTOS

- Una **clase abstracta** sirve como una clase base para herencia, por lo que no se puede instanciar.
- Un **método abstracto** no tiene cuerpo (sólo firma) y **debe ser sobrescrito** o implementado (con la misma firma) en la clase derivada.

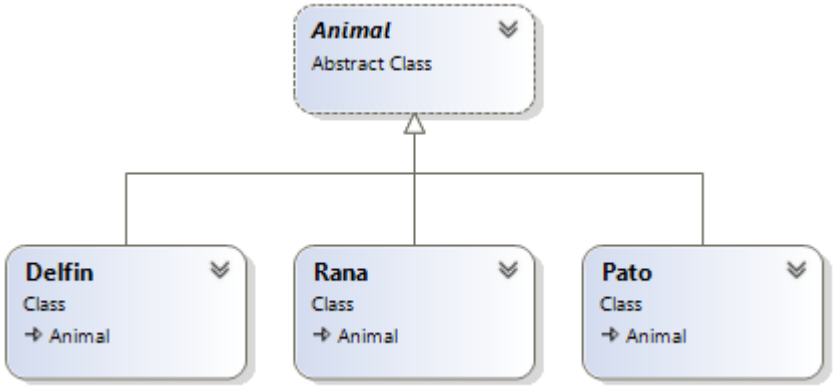
- Ejemplo: clase base Animal




```
abstract class Animal
{
    public WindowsMediaPlayer wplayer;
    private int x;
    private int y;

    public Animal(int x, int y)
    {
        this.x = x;
        this.y = y;
        wplayer = new WindowsMediaPlayer();
    }

    public virtual void Moverse()...

    public abstract void HacerSonido();
}
```

CLASES Y MÉTODOS ABSTRACTOS (CONT..)

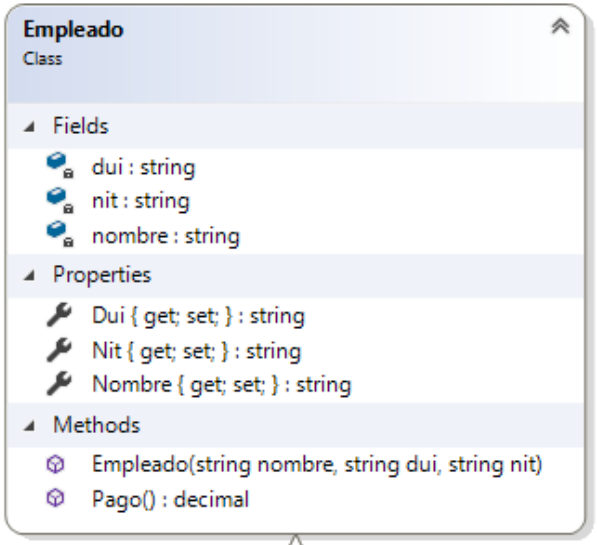
- **Ejemplo**: método abstracto HacerSonido, heredado de la clase base Animal y obligatoriamente sobrescrito (implementado) en la clase derivada Pato.



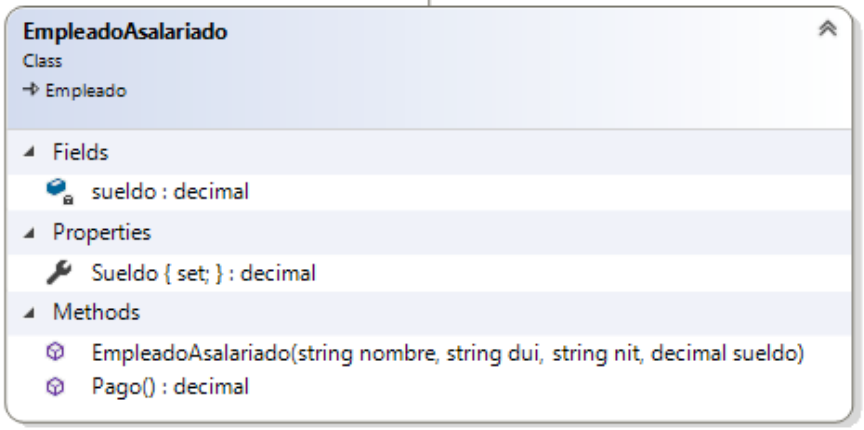

```
class Pato : Animal
{
    public Pato(int x, int y) ...

    public override void Moverse()...

    public override void HacerSonido()
    {
        wplayer.URL = AppDomain.CurrentDomain.BaseDirectory + "/Sonidos/pato.mp3";
        wplayer.controls.play();
    }
}
```

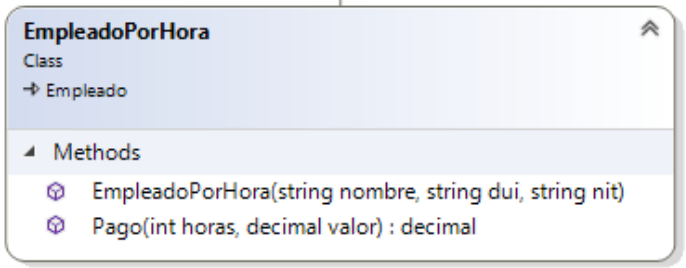
DEMO: EMPLEADOS (POLIMORFISMO)

- Clase base



- La clase derivada EmpleadoAsalariado, **sobrescribe** (override) el método Pago de la clase base Empleado

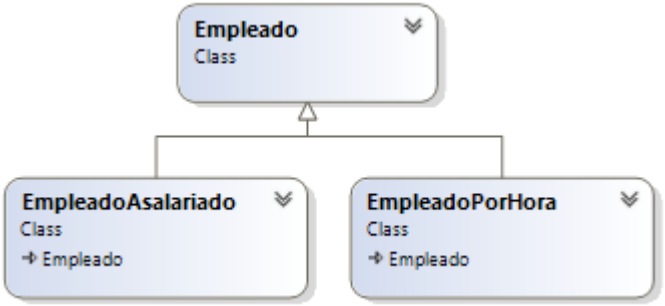




- La clase derivada EmpleadoPorHora, **sobrecarga** (overload) el método Pago de la clase base Empleado

DEMO: EMPLEADOS (CONT.)

- **Clase base** Empleado

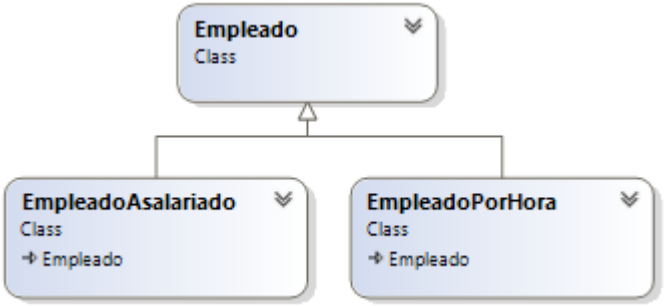



```
class Empleado
{
    //campos
    private string nombre;
    private string dui;
    private string nit;

    //método constructor
    public Empleado(string nombre, string dui, string nit)
    {
        this.nombre = nombre;
        this.dui = dui;
        this.nit = nit;
    }

    //propiedades
    public string Nombre { get => nombre; set => nombre = value; }
    public string Dui { get => dui; set => dui = value; }
    public string Nit { get => nit; set => nit = value; }

    //método virtual
    public virtual decimal Pago()
    {
        return 300;
    }
}
```

La clase base posee el método (**virtual**) Pago, el cual puede ser sobrescrito por las clases derivadas

DEMO: EMPLEADOS (CONT.)

- **Clase derivada** EmpleadoAsalariado

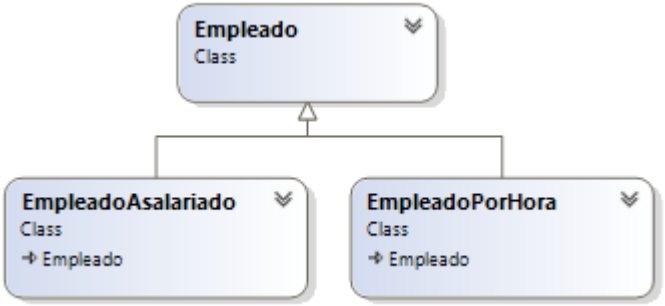



```
class EmpleadoAsalariado : Empleado
{
    //campo
    private decimal sueldo;

    //método constructor
    public EmpleadoAsalariado(string nombre, string dui, string nit, decimal sueldo)
        : base(nombre, dui, nit)
    {
        this.sueldo = sueldo;
    }

    //propiedad
    public decimal Sueldo { set => sueldo = value; }

    //sobreescritura del método Pago heredado de la clase base
    public override decimal Pago()
    {
        return sueldo - (sueldo * 0.1m);
    }
}
```

El método pago, heredado de la clase base, fue sobrescrito (**override**) en la clase derivada

DEMO: EMPLEADOS (CONT.)

- **Clase derivada** EmpleadoPorHora




```
class EmpleadoPorHora : Empleado
{
    //método constructor
    public EmpleadoPorHora(string nombre, string dui, string nit)
        : base(nombre, dui, nit)
    {
    }

    //sobrecarga del método Pago heredado de la clase base
    public decimal Pago(int horas, decimal valor)
    {
        return horas * valor;
    }
}
```

El método Pago, fue sobrecargado (**overload**) en la clase derivada, dado que el método pago se creó con una nueva firma

BIBLIOGRAFÍA

- Asad, A. (2017). The C# Programmer's Study Guide (MCSD) Exam: 70-483. Pakistan: Apress.
- Ceballos, F. (2013). Microsoft C# Curso de programación. Segunda edición. México: Alfaomega.
- Putier, S. (2018). C# 7 y Visual Studio 2017. España: ENI.

- Microsoft Docs:
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/polymorphism
- https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/object-oriented-programming