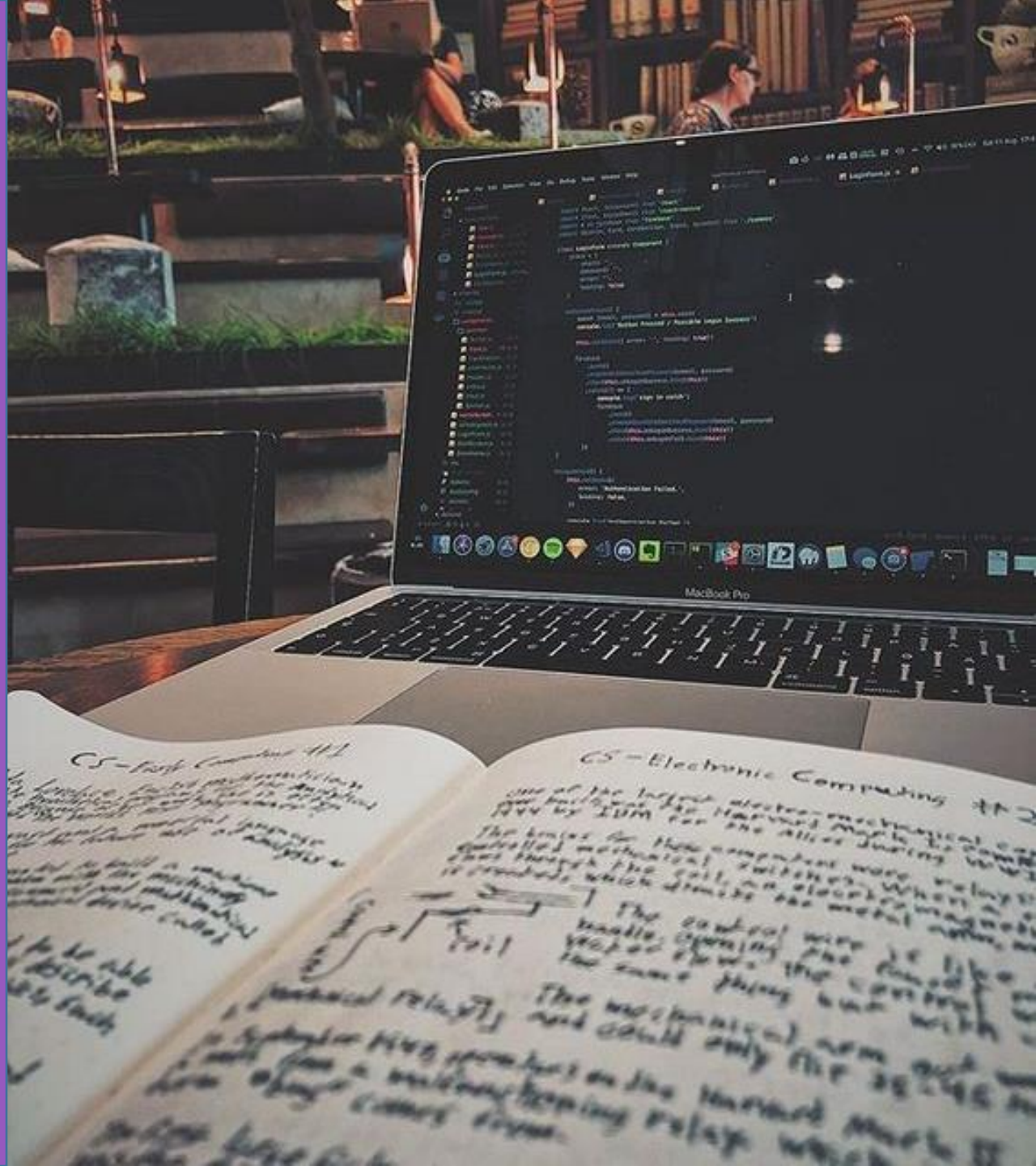


# Clase Nro 3

## P00 segunda parte

- Herencia
- Sobrecarga
- Polimorfismo



# Clase Abstractas

## ¿Que es una clase abstracta?

Una clase abstracta es aquella que forzosamente se debe **derivar** si se desea que se puedan crear objetos de la misma o acceder a sus miembros. Es decir, son clases de las que **no es posible crear instancias**.

Para declarar una clase abstracta, utilizamos el modificar **abstract**.

```
public abstract class ClaseAbstracta
{

}
```

# Clase Abstractas

## Propiedades de las clases **Abstractas**

Podemos declarar una clase o un miembro de forma abstracta utilizamos el modificador **abstract**.

**Teniendo en cuenta lo siguiente:**

1. Si algún método de la clase es abstracto, entonces la clase debe ser declarada abstracta.
2. Las propiedades y métodos declarados como abstractos NO proporcionan implementación; solo declaraciones.
3. Una clase abstracta puede también tener propiedades o métodos no abstractos.
4. Una clase abstracta puede tener atributos comunes.
5. Los métodos abstractos serán sobrescritos en las subclases. Si una subclase no implementa un método abstracto de la superclase tiene un método no ejecutable, lo que la obligaría a ser una subclase abstracta.

```
public abstract class ClaseAbstracta
{
    private int atributo;
    public double MetodoConcreto() {return 0;}
    public abstract double MetodoAbstracto();
}
```

# Clase Abstractas

## Ejemplo de implementación

- Método abstracto (debe ser implementado en las clases derivadas)
- Los elementos abstractos DEBEN ser sobrescritos en la subclase.
- En la subclase, se utiliza “override” para realizar la implementación correspondiente.

## Beneficios

- Reutilización
- Polimorfismo

```
public abstract class FiguraGeometrica
{
    public abstract double Area();
    public abstract double Perimetro();
}

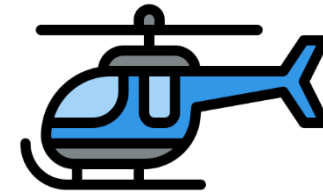
public class Cuadrado : FiguraGeometrica
{
    private double Lado;
    public override double Area(){
        return Lado * Lado;
    }
    public override double Perimetro(){
        return Lado * 4;
    }
}

public class Rectangulo : FiguraGeometrica
{
    private double b,h;
    public override double Area(){
        return (b * h)/2;
    }
    public override double Perimetro(){
        return (b + h)/2;
    }
    [...]
}
[...]
```

```
FiguraGeometrica MiCuadrado = new Cuadrado(3);
FiguraGeometrica MiRectangulo = new Rectangulo(3,5);
MiCuadrado.Area();
MiRectangulo.Area();
```

# Ejercicio: Simulando construcción de vehículos

- EncenderMotor(string sonido),
- Conducir()
- ApagarMotor(string sonido)
- Volar()



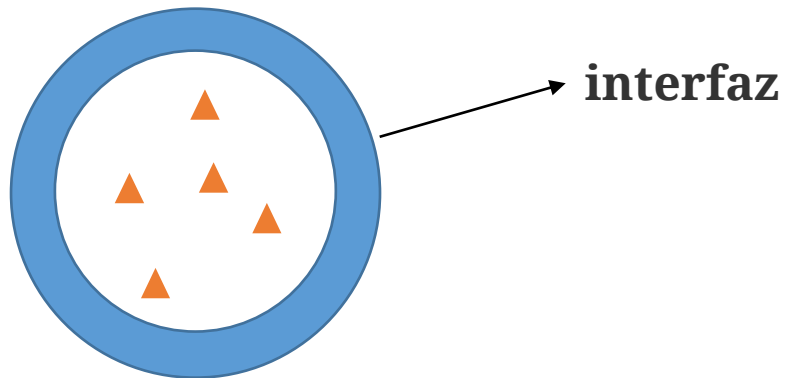
Tiene motor	😊	No Tiene Motor	❌	Tiene Motor	😊
Conduce	😊	Se puede Conducir	😊	Se Puede conducir	😊
No Puede volar	❌	No puede volar	❌	Puede volar	😊

# Interfases

¿Qué es la interfaz de una clase?

En teoría de orientación a objetos, la interfaz de una clase es todo lo que podemos hacer con ella. A efectos prácticos: todos los métodos, propiedades públicas de la clase conforman su interfaz.

**Toda clase tiene una interfaz** que define que podemos hacer con los objetos de dicha clase.



Tanto las clases abstractas como las interfaces son mecanismos que obligan la herencia

# Interfases

## Interfases en C#

Las **interfases** son contratos que obligan a las clases que las implementan a implementar los miembros definidos en el cuerpo de la **interfaz**

Mediante las interfases puede incluir, por ejemplo, un comportamiento de varios orígenes a una misma clase.

Una declaración de interfaz puede contener declaraciones (firmas sin ninguna implementación) de los miembros siguientes:

- Métodos
- Propiedades
- Indexadores
- Eventos

Nota:

A partir de C# 8.0, una interfaz puede definir una implementación predeterminada de miembros.

# Interfaces

Las Interfaces nos permiten definir un "contrato" sobre el que podemos estar seguros de que, las clases que las implementen lo van a cumplir.

## Beneficios

- Herencia múltiple
- Polimorfismo

```
Public Interface IAnimal
{
    void Sonido();
}

Public Interface Ivolador
{
    void Volar();
}

Public class Pato : Ianimal, IVolador
{
    public void Sonido(){
        Console.WriteLine("Cuack");
    }

    public void Volar(){
        Console.WriteLine("Volando");
    }
}

Public class PatoGoma : IAnimal
{
    public void Sonido(){
        Console.WirteLine("Sqcuchin scuchin!");
    }
}
[...]
```

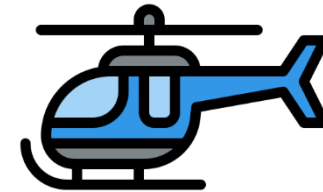
  

```
IAnimal patito = new Pato(3);
IAnimal patitoGoma = new PatoGoma(3,5);
patito.Sonido();
patitoGoma.Sonido();
```



# Ejercicio: Simulando construcción de vehículos

- EncenderMotor(string sonido),
- Conducir()
- ApagarMotor(string sonido)
- Volar()



Tiene motor	😊	No Tiene Motor	❌	Tiene Motor	😊
Conduce	😊	Se puede Conducir	😊	Se Puede conducir	😊
No Puede volar	❌	No puede volar	❌	Puede volar	😊