UTN F r a
Avellaneda

Programación y Laboratorio II

Clase 08

Herencia

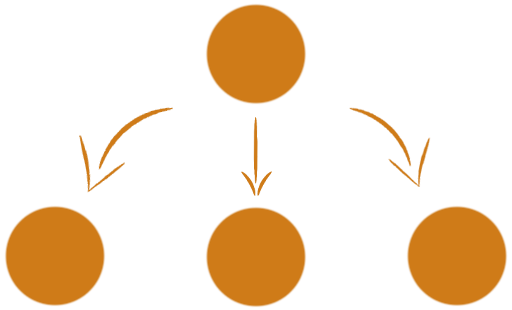
Temario

Herencia

- ¿Qué es la herencia?
- Tipos de herencia
- Razones de utilizar la herencia
- Clases bases y derivadas
- Principio de Sustitución de Liskov

Herencia en C#

- Implementación
- Herencia en la clase base System.Windows.Forms
- Constructores
- Clases selladas
- Modificador protected



01.

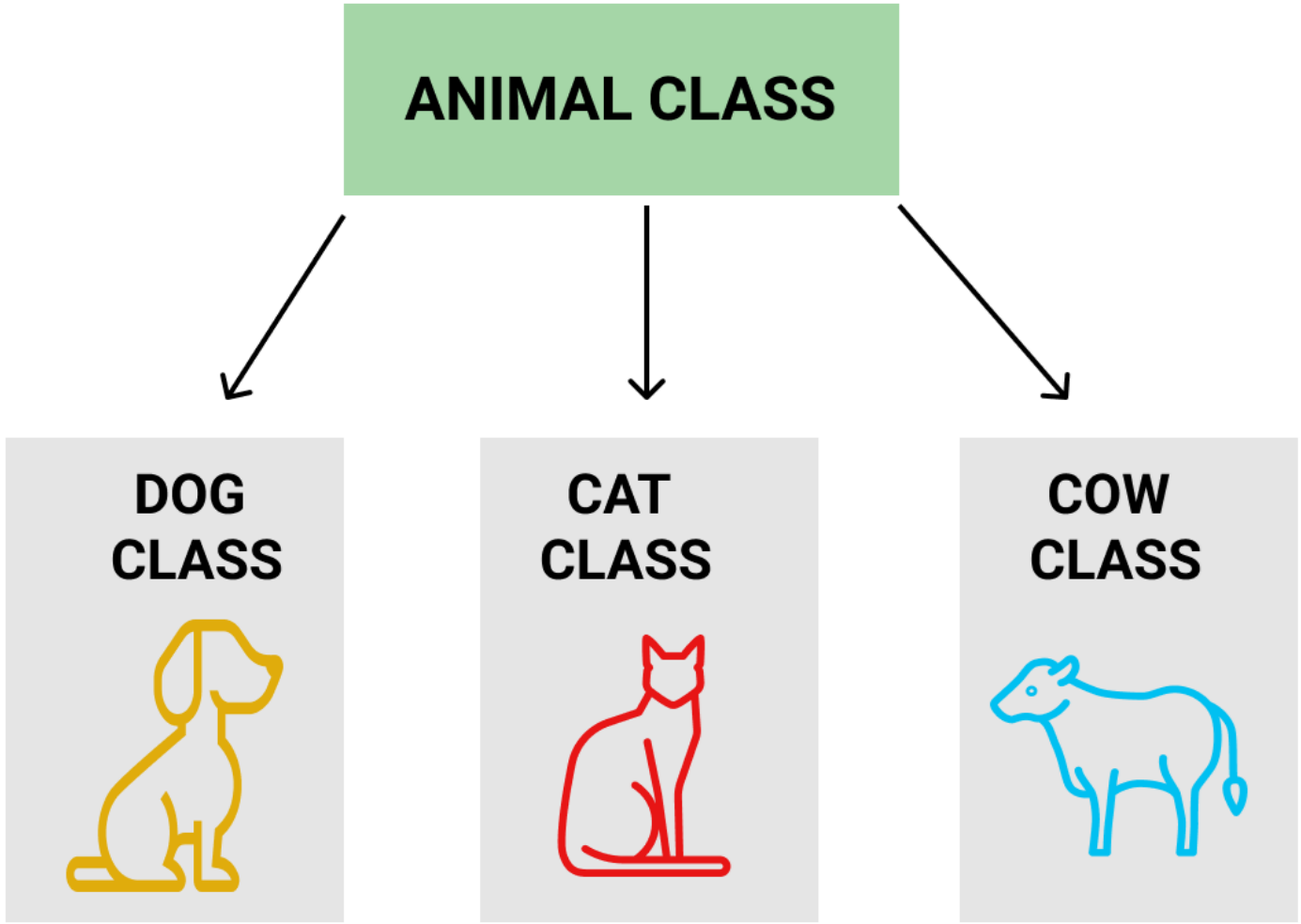
Herencia

¿Qué es la Herencia?

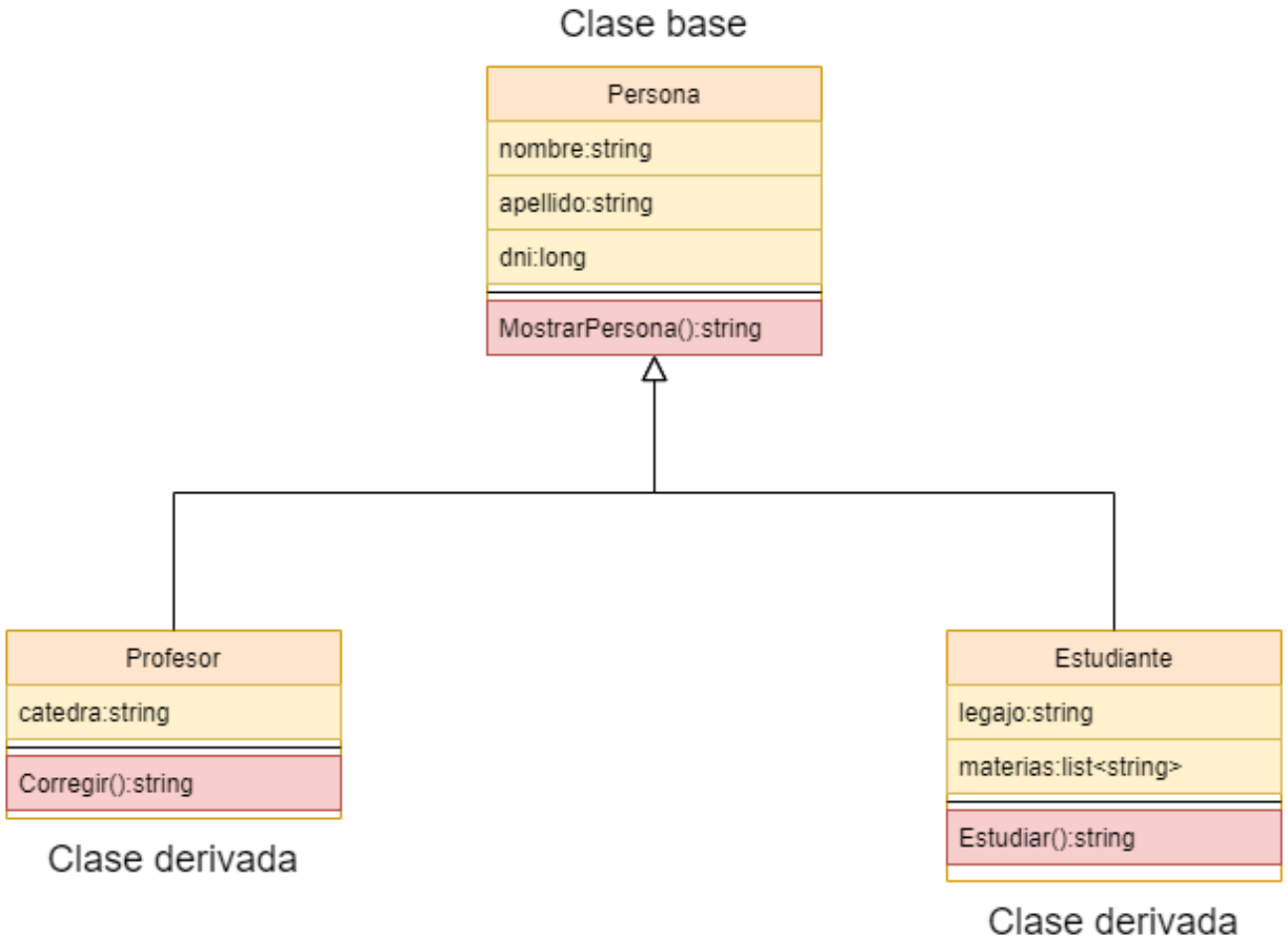
La herencia es uno de los cuatro pilares de la programación orientada a objetos.

Es una relación entre una o más clases en la que se comparte el comportamiento y funcionalidad definido en otra clase.

Que es la herencia?



Clases bases y derivadas



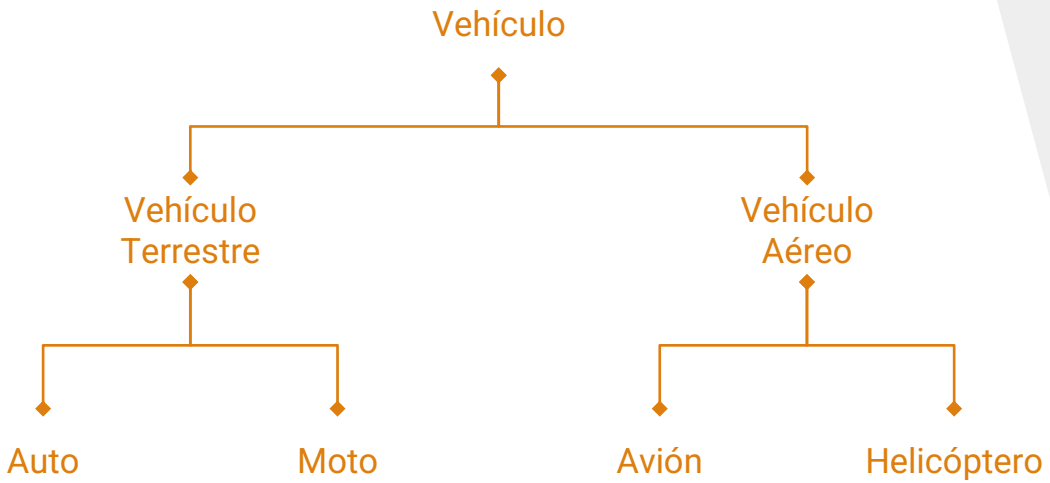
La clase principal en la que se basaran las siguientes clases se la conoce como **clase base**, mientras que la subclase de esta se la conoce como **clase derivada.**

Una **clase derivada** posee los atributos y métodos de la **clase base** además de los propios.

En este ejemplo tenemos la clase persona que tiene dos **clases derivadas**, Profesor y estudiante además de tener sus propios miembros, **posee los miembros heredados de persona.**

Razones de utilizar
Herencia

La herencia nos permite llevar la generalidad de una clase a una especialización en base a clases ya existentes en lugar de crear nuevas clases desde cero.

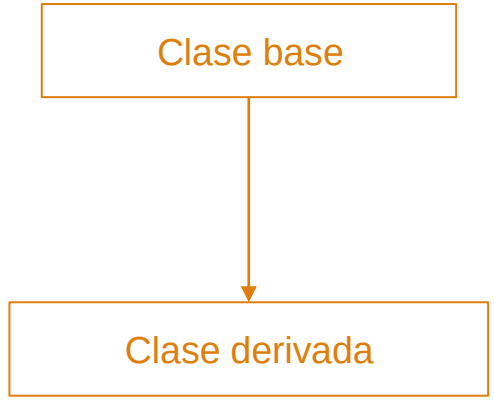


Tipos de herencia

Herencia Simple

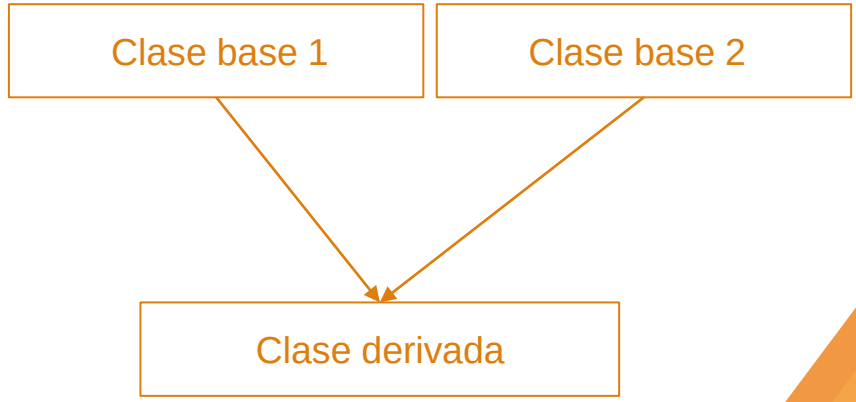
Una clase derivada **sólo puede heredar de una clase base.**

El framework de .NET solo soporta este tipo de herencia.



Herencia Múltiple

Una clase derivada **puede heredar de una o más clases base.**



Principio de Sustitución de Liskov

Cada clase que hereda de otra puede usarse como su padre sin necesidad de conocer las diferencias entre ellas.

Si **S** es un subtipo de **T**, entonces los objetos de tipo **T** en un programa de computadora pueden ser sustituidos por objetos de tipo **S.**

T

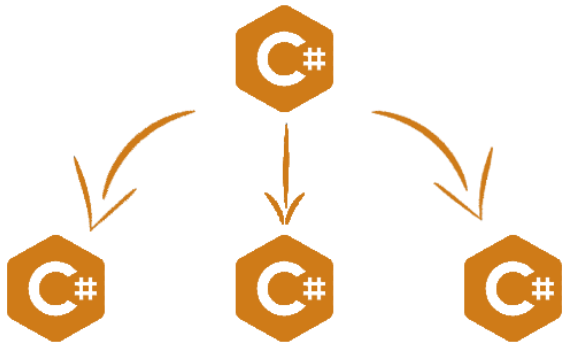
S

```
T objeto = new S();
```

```
Animal animal = new Perro();
Persona persona = new Alumno();
```

Barbara Liskov

Barbara Liskov, reconocida ingeniera de software que fue la primera mujer de los Estados Unidos en conseguir un doctorado en Ciencias de la Computación, ganadora de un premio Turing y nombrada doctora honoris causa por la UPM.



02.

Herencia en C#

Implementación

```
public class Estudiante : Persona
{
    private int legajo;
    private List<string> materias;

    public string Estudiar()
    {
        return "Estudiando....";
    }

}
```

Para implementar **herencia** en una **clase derivada** lo único que tenemos que hacer es al final de la declaración de la clase debemos darle el nombre de la **clase base** precedido por el operador (:).

Implementación

Puntos importantes

- Las clases derivadas heredan todos los miembros de la clase base excepto los constructores.
- La accesibilidad de una clase derivada no puede ser mayor a la de su clase base, por ej una clase pública no puede heredar de una clase privada.
- Los miembros públicos de una clase base implícitamente se convierten en miembros públicos de la clase derivada.
- Aunque una clase herede todos los miembros si estos son privados, solo la clase base tiene acceso a ellos, **aunque la clase derivada también los hereda.**
- Por transitividad, una clase C que hereda de una clase B que a su vez hereda de una clase A, también hereda de la clase A

Herencia en la clase System.Windows.Forms

UTN Fra Avellaneda

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

}
```

```
public class Form : ContainerControl
{

    public Form();

    public static Form ActiveForm { get; }

    public bool AutoScale { get; set; }

    public Point Location { get; set; }

    public bool KeyPreview { get; set; }

    public bool IsRestrictedWindow { get; }

    public bool IsMdiContainer { get; set; }

    public bool IsMdiChild { get; }

    public Icon Icon { get; set; }

    public bool HelpButton { get; set; }

    public DialogResult DialogResult { get; set; }

    public Point DesktopLocation { get; set; }
```

Constructores

Si la clase base sobrescribe su constructor por defecto tenemos que llamarlo desde la clase derivada.

```
public Estudiante(int legajo, List<string> materias,string nombre):base(nombre)
{
    this.legajo = legajo;
    this.materias = materias;
}
```

Hereda de →

```
public Persona(string nombre)
{
    this.nombre = nombre;
}
```

Para hacer esto utilizaremos la palabra dedicada **base** precedida del operador (:) con todos los argumentos del constructor de la clase base.

Si el constructor de la clase base está sobrecargado de puede llamar a cualquiera de estos. →

```
public Persona(string nombre)
{
    this.nombre = nombre;
}
public Persona(long dni)
{
    this.dni = dni;
}
```

Clases selladas

No todas las clases están diseñadas para que otras deriven de ellas y C# nos permite crear un tipo de clase llamado **sealed**(sellada) y lo que esto logra es evitar que esa clase puede tener derivadas.

```
public sealed class PersonaSellada
{
}
```

Declaración de una clase **sellada.**

Modificador Protected

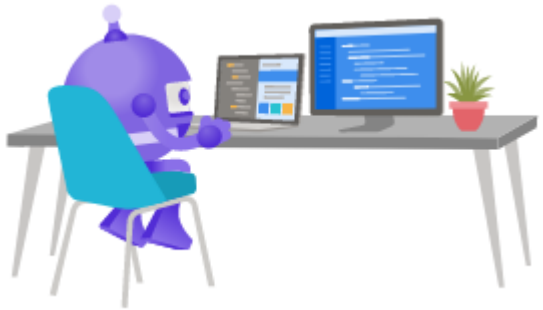
Cuando estamos frente a una relación de herencia el modificador de visibilidad protected nos ayuda a **extender la visibilidad de nuestros miembros heredados mientras al mismo tiempo no romper el encapsulamiento de nuestras clases**.

Esto quiere decir que para cualquier clase afuera de nuestra relación de herencia (base - derivada) estos miembros se van a comportar como privados, pero para nuestras clases derivadas se van a comportar como miembros públicos.

```
public class Persona
{
    protected string nombre;
    protected long dni;
}
```

Declaración de atributos protected en una clase base.

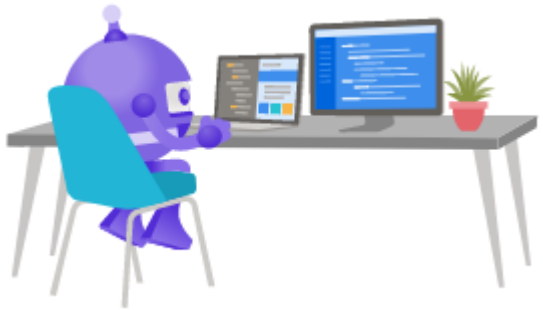
Ejercicios



- I01 - El viajar es un placer
- I02 - La centralita: Episodio I

https://codeutnfra.github.io/programacion_2_laboratorio_2_apuntes/

Tarea



- I01 - Herencia deportiva
- C02 - Go Speed Racer Go!

https://codeutnfra.github.io/programacion_2_laboratorio_2_apuntes/