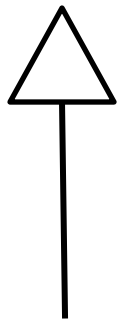
Herencia

Herencia

Es un tipo especial de relación, representa una relación ***es un/una*** entre clases.

La clase de la cual se hereda se llama **clase base** o superclase y la clase que hereda se llama **derivada** o subclase. Se representa con un triángulo blanco en dirección a la clase base.



Los atributos y métodos de la clase base (salvo los privados) son también atributos y métodos de la clase derivada, pero en el diagrama ***no*** se incluyen nuevamente en la clase derivada.

Herencia

El concepto de herencia es muy importante en Orientación a Objetos porque:

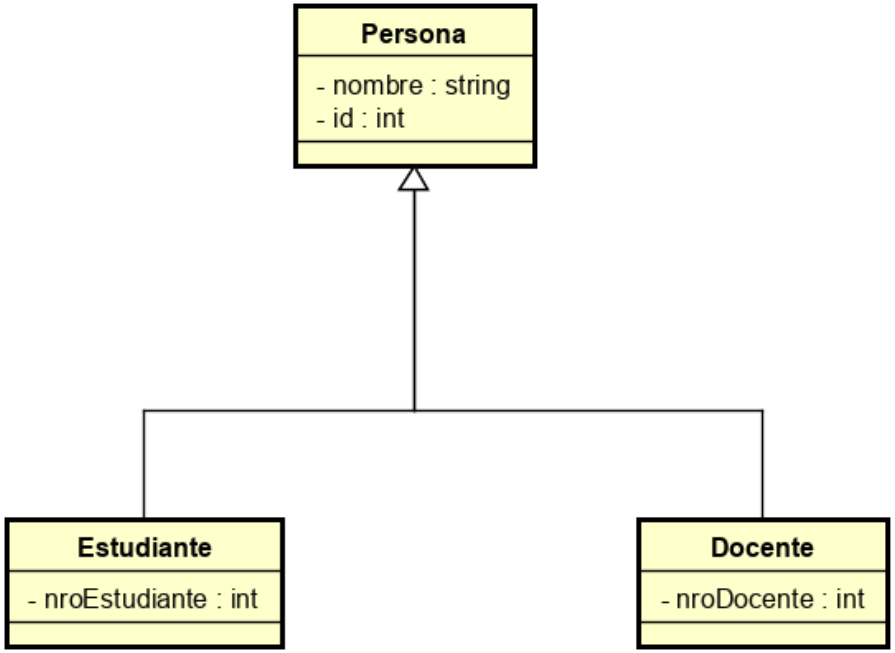
Al usar este concepto agrupamos características que lógicamente tienen sentido juntas

Reducimos la cantidad de atributos y métodos que debemos definir para nuestras clases

Cuando llega la hora de implementar, las clases ***derivadas*** no necesitan definir nuevamente los atributos y métodos de su clase base, sino que ***heredan*** los de su clase base

Lo anterior es ventajoso porque es una forma de ***reutilización de código*** ya escrito. En la clase derivada sólo debemos definir los miembros propios de dicha clase. Los de su clase base son heredados (ya implementados)

Herencia


Persona
- nombre : string
- id : int
Estudiante
- nroEstudiante : int
Docente
- nroDocente : int

Herencia

El Estudiante tiene el atributo nroEstudiante y hereda de la clase base el nombre y el id a través de sus propiedades que son públicas (no directamente porque el atributo se define como privado)

De igual forma el Docente tiene el atributo nroDocente y hereda de la clase base nombre y el id a través de sus propiedades que son públicas (no directamente porque el atributo se define como privado)

El constructor de la clase derivada recibe todos los parámetros y pasa al constructor de la clase base lo que correspondan.

Herencia clase base

```
public class Persona
{
    private int _id;
    private string _nombre;

    public int Id
    {
        get { return _id; }
    }

    public string Nombre
    {
        get { return _nombre; }
    }

    public Persona(int id, string nombre)
    {
        _id = id;
        _nombre = nombre;
    }
}
```

Herencia clase derivada

```
// se indica que hereda de la clase Persona
public class Docente : Persona
{
    private int _nroDocente;

    public int NroDocente
    {
        get { return _nroDocente; }
    }

    // se reciben todos los parámetros
    public Docente(int nroDocente, int id, string nombre) : base(id, nombre)
    {
        _nroDocente = nroDocente;
    }
}
```

Herencia

Para detectar la presencia del concepto de herencia en nuestro dominio podemos aplicar dos técnicas :

Generalización : Si en el dominio de nuestro problema detectamos clases que poseen características en común, podemos "quitarlas" de dichas clases y crear una nueva clase base que las contenga (siempre y cuando el resultado tenga sentido).

Especialización : Si ya tenemos herencia en nuestro diagrama y creamos una clase cuyas características se asemejan a las de la herencia, podemos incluir la nueva clase como derivada en dicha herencia (siempre y cuando el resultado tenga sentido). En la nueva clase incluimos sólo las características que ***no*** están en la clase base.

Instancias de clases

```
// definición de la lista
private List<Persona> _personas = new List<Persona>();

private void PrecargaEjemplo()
{
    Persona unaP1 = new Persona(1, "Ana");
    Persona unaP2 = new Persona(2, "Juan");
    Docente unD1 = new Docente(123123, 3, "Docente 1");
    Docente unD2 = new Docente(567890, 4, "Docente 2");

    _personas.Add(unaP1);
    _personas.Add(unaP2);
    _personas.Add(unD1);
    _personas.Add(unD2);
}
```

```
public string ListarPersonas()
{
    string retorno = "";
    foreach (Persona p in _personas)
    {
        // se usa el método ToString
        retorno += p;
    }
    return retorno;
}
```

Clases Abstractas

Una clase abstracta es, en contraposición con una clase concreta, una clase base de la cual se debe heredar (en C# se declaran como **abstract**).

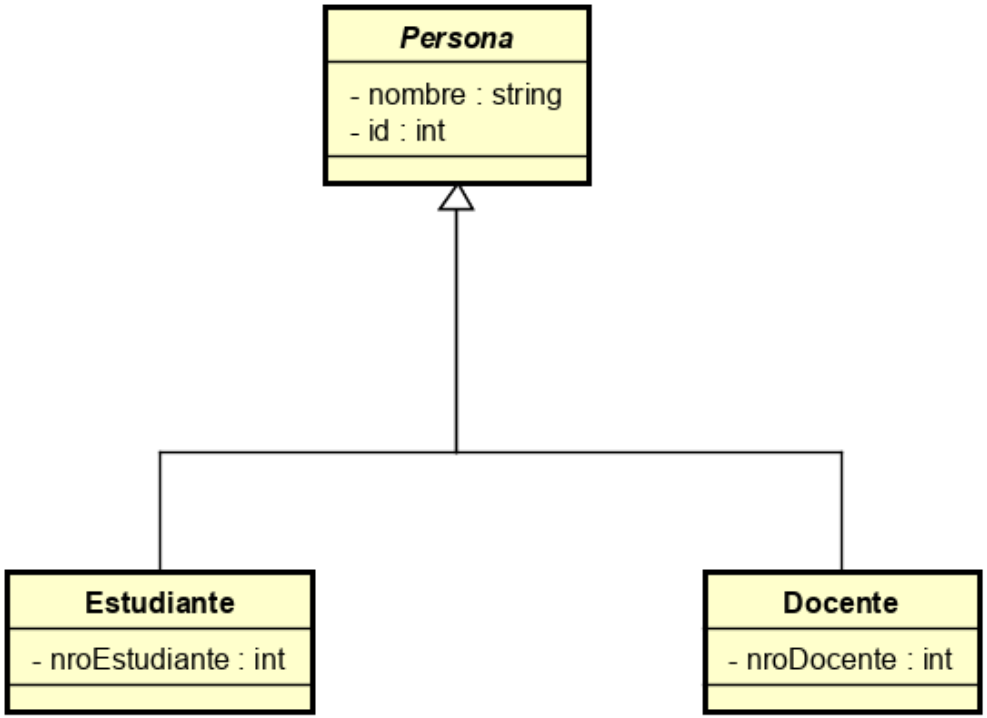
No se crean instancias de una clase abstracta.

Son de gran utilidad, no es razonable instanciar. Ej.: persona, cuenta bancaria.

Clases Abstractas

Una clase abstracta puede ser totalmente funcional (completamente implementada) o simplemente puede definir los miembros que deben obligatoriamente implementar las clases derivadas.

Clases Abstractas



Clases Abstractas

Sintaxis:

Para que una clase sea considerada abstracta, se debe preceder de la palabra clave abstract.

public abstract class Persona.

Los miembros abstractos son solo cuando no contienen código, solamente la firma. Es obligatorio que sea sobrescrito por las clases derivadas, y que estas lo implementen.

Clases Abstractas

```
public abstract class Persona
{
    private int _id;
    private string _nombre;

    public abstract string DarIdentidad();
```

Clases Abstractas

Si se declara uno de los miembros de una clase como abstracto, se debe declarar como abstracta la clase. En caso contrario, se producirá un error de compilación.

Los miembros abstractos no se implementan en la clase abstracta, sino en las clases que se derivan de ella.

Clases Abstractas

Si una clase se declara como abstracta, no es necesario declarar métodos abstractos en ella.

En las clases derivadas es necesario seguir utilizando override.