

Preguntas detonadoras



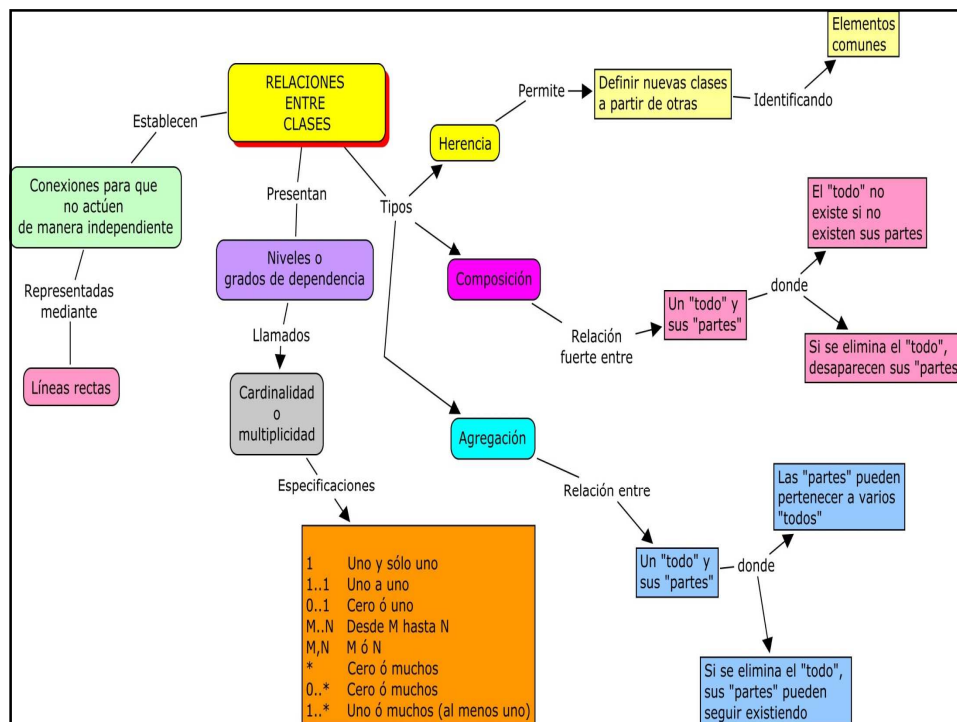
- ❑ ¿Qué ventajas ofrece la herencia a un programador?
- ❑ ¿Cuál es la diferencia entre herencia simple y herencia múltiple?
- ❑ Si una clase recibe herencia de una clase y varias interfaces, ¿se considera herencia múltiple?
- ❑ Si una clase transmite (hereda) sus componentes a dos o más clases, ¿se considera herencia múltiple?
- ❑ ¿Se pueden diseñar aplicaciones con herencia múltiple en C# .NET?
- ❑ Si una clase abstracta no puede generar objetos, ¿entonces para qué sirve?
- ❑ ¿Se puede modificar la implementación de un método heredado?
- ❑ Si un miembro abstracto no tiene implementación, ¿entonces para qué sirve?
- ❑ En una clase abstracta, ¿todos sus miembros son abstractos?
- ❑ ¿Cuál es la ventaja de sobrescribir el método ToString()?
- ❑ ¿Para qué sirve una clase sellada (sealed)?
- ❑ ¿En qué se parece una interfase a una clase abstracta? ¿En qué difieren?

3

Relaciones entre clases:

Herencia, Composición y
Agregación

4



Herencia

- **Característica de la POO que permite definir nuevas clases a partir de otras ya existentes.**
- **Las clases existentes “transmiten” sus características.**

Herencia (cont.)

- **Puede usarse para:**
 - Relaciones del tipo “es un”
 - Ejemplo: Un Gerente “es un” Empleado con características propias adicionales.
- **Objetivo: Reutilización de código.**

7

Ejercicio

- **Se deben modelar dos clases con las siguientes características:**

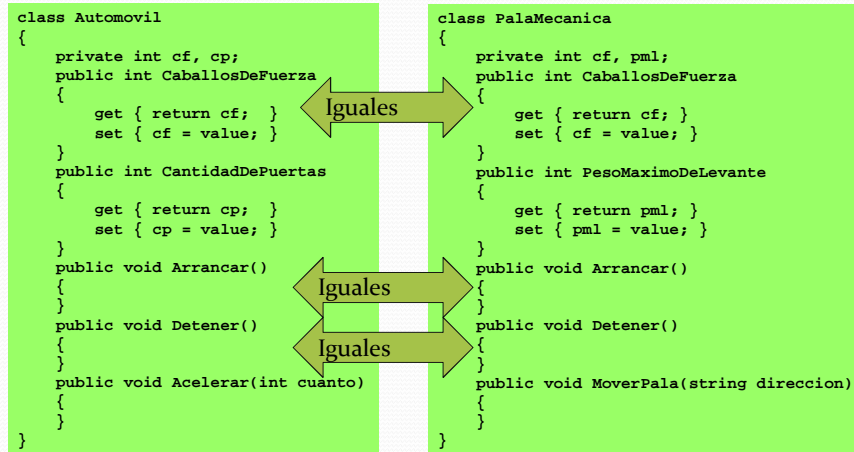
Automovil
CaballosDeFuerza: int CantidadDePuertas: int
Arrancar() : void Detener() : void Acelerar(int cuanto): void

PalaMecanica
CaballosDeFuerza: int PesoMaximoDeLevante: int
Arrancar() : void Detener() : void MoverPala(string direccion) : void

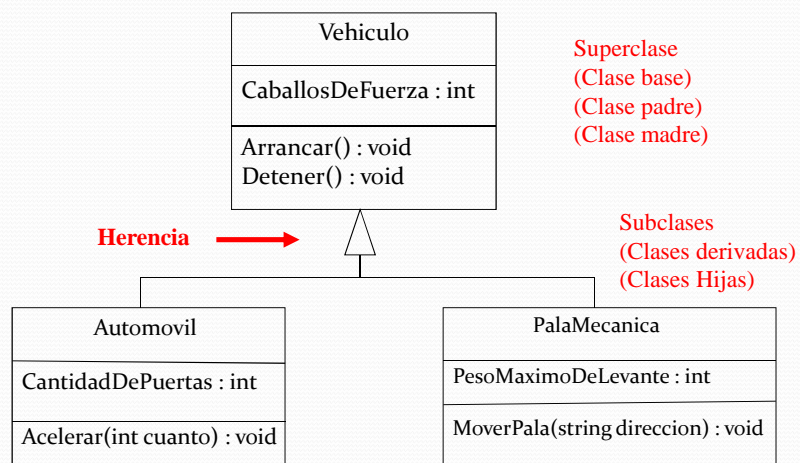
8

Mal diseño (no recomendable)

- Modelarlas de manera independiente.



Diseño usando herencia (recomendado)



Definición de las clases usando herencia en C#

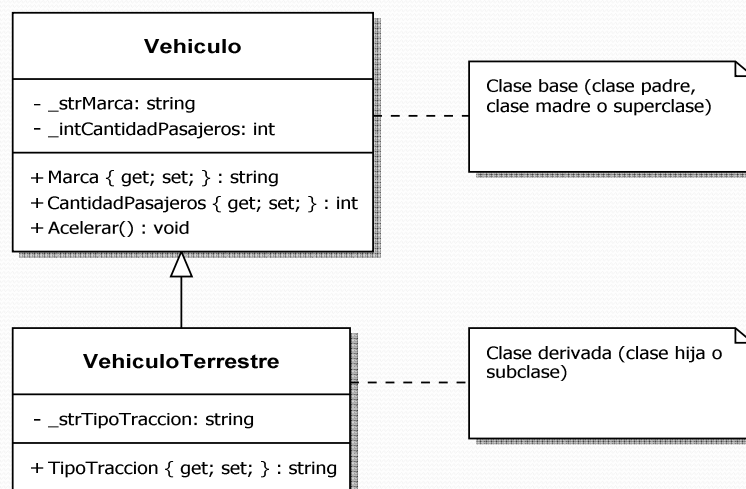
```
class Vehiculo
{
    private int cf;
    public int CaballosDeFuerza
    {
        get { return cf; }
        set { cf = value; }
    }
    public void Arrancar()
    {
    }
    public void Detener()
    {
    }
}
```

```
class Automovil : Vehiculo
{
    private int cp;
    public int CantidadDePuertas
    {
        get { return cp; }
        set { cp = value; }
    }
    public void Acelerar(int cuanto)
    {
    }
}
```

```
class PalaMecanica : Vehiculo
{
    private int pml;
    public int PesoMaximoDeLevante
    {
        get { return pml; }
        set { pml = value; }
    }
    public void MoverPala(string direccion)
    {
    }
}
```

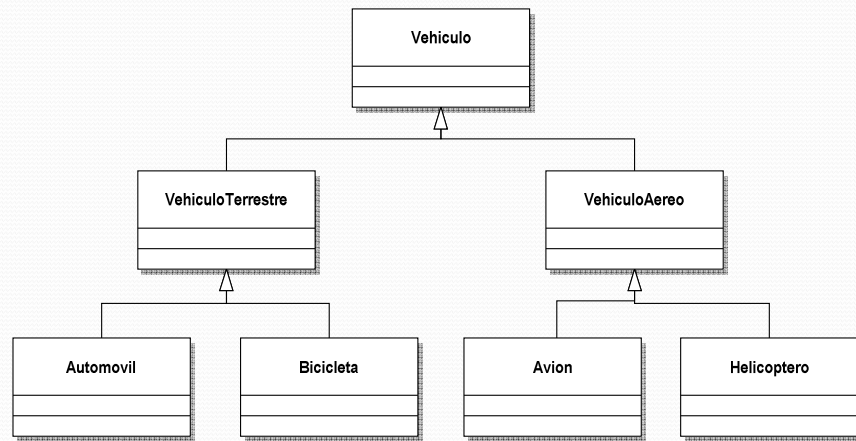
11

Ejemplo de herencia



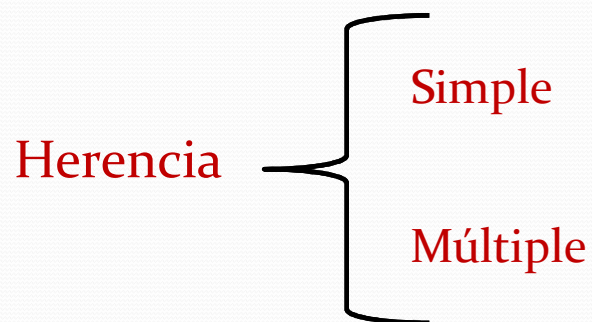
12

Ejemplo de herencia con varios niveles

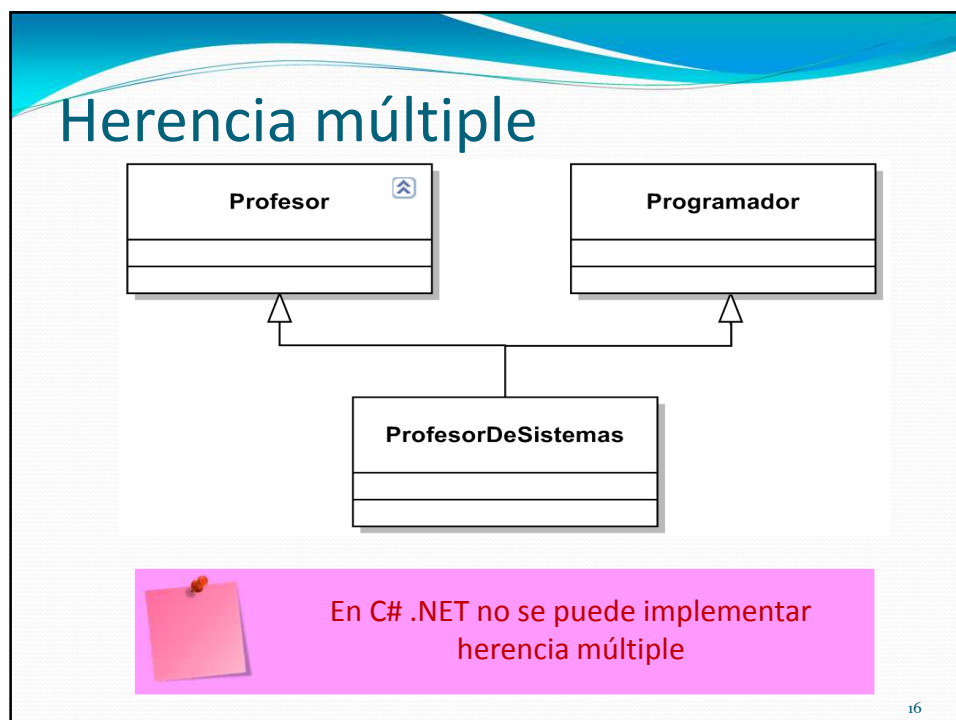
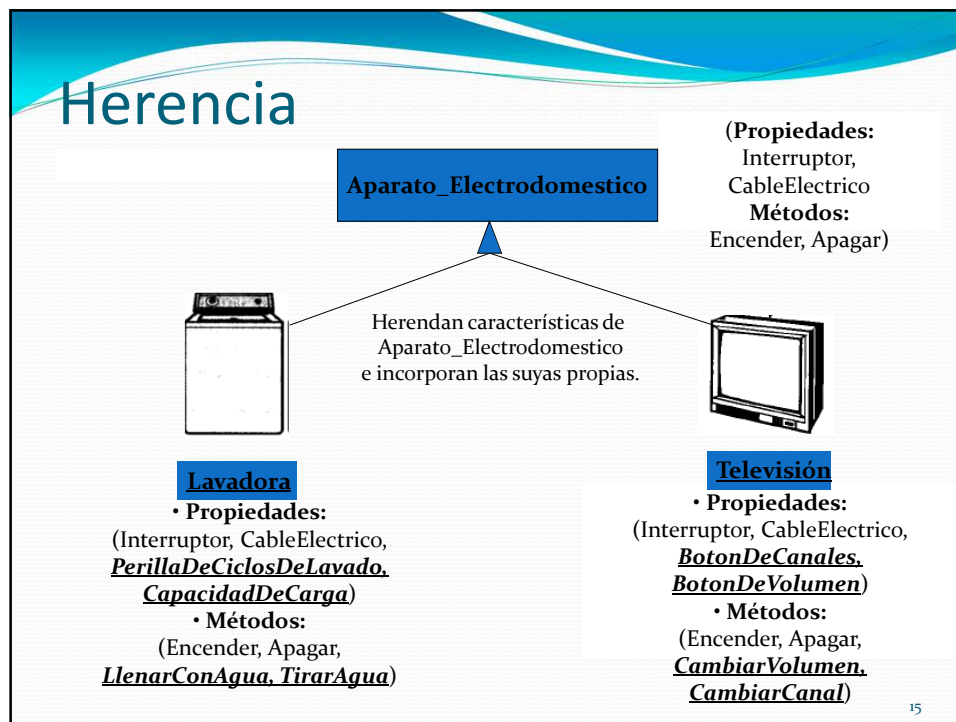


13

Tipos de herencia



14



Herencia en C#

- En C# solo se permite Herencia simple.
- Ejemplo de Herencia en C#

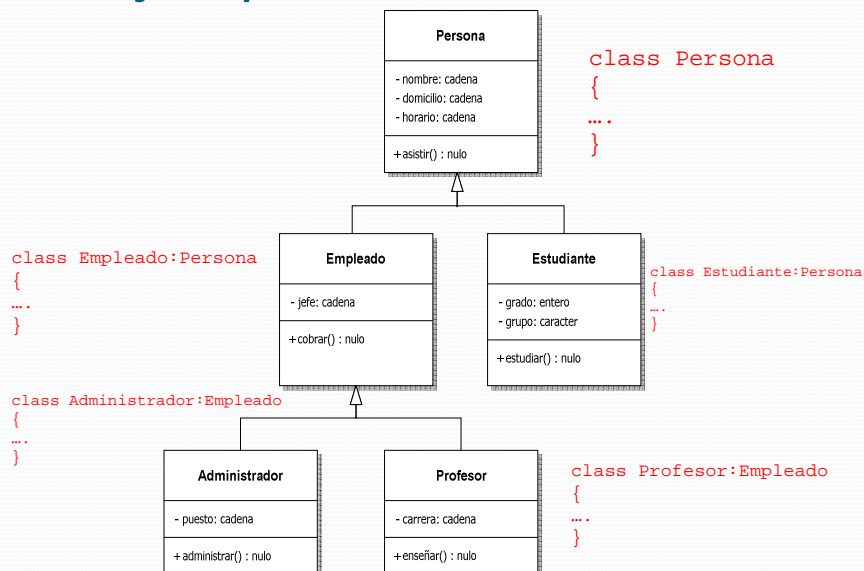
```
class A
{
}
class B : A
{
}
```

Indica que B "Hereda de" A

- Todos los objetos heredan de `System.Object`

17

Otro ejemplo de herencia



18

Uso de la Herencia

- **Beneficios:**
 - Permite escribir menos código.
 - Mejora la reusabilidad de los componentes.
 - Facilita el mantenimiento del sistema completo.
- **Útil para un buen diseño del programa.**
- **Un diseño pobre sin herencia implementaría las clases involucradas de manera independiente.**

19

Ejercicio

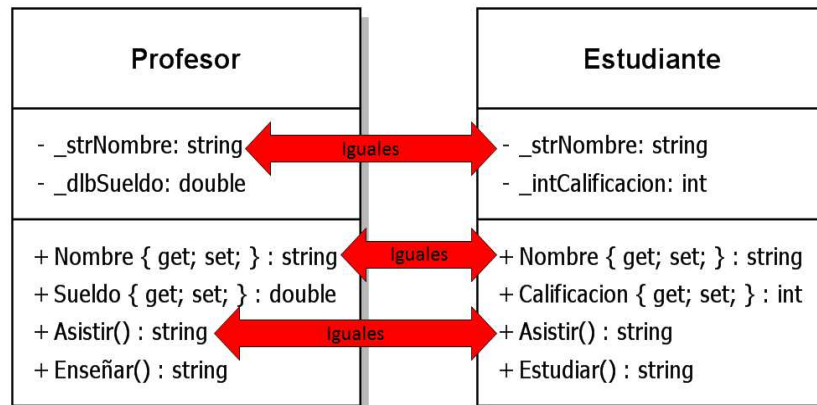
- **Una escuela desea modelar los datos y las actividades de sus profesores y de sus estudiantes.**

PROFESOR	
Datos	Actividades
Nombre (cadena)	Asistir a la escuela
Sueldo (numérico real)	Enseñar

ESTUDIANTE	
Datos	Actividades
Nombre (cadena)	Asistir a la escuela
Calificación (numérico entero)	Estudiar

20

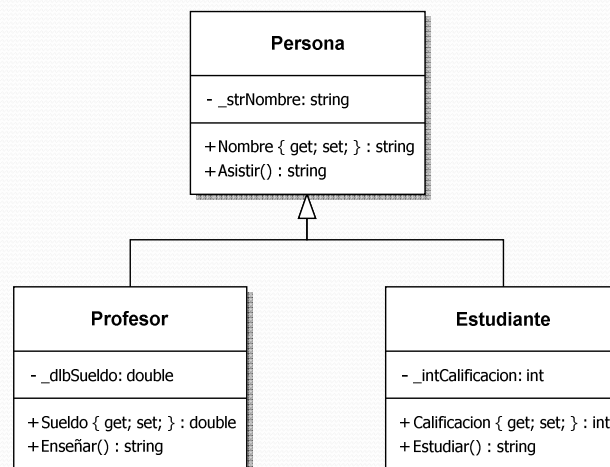
Mal diseño (no recomendable)



21

Diseño con herencia

- Prog. 5.1.- HerenciaConsola



22

Diseño con herencia

- El Prog. 5.2.- HerenciaFormas utiliza el mismo diseño de herencia del proyecto de consola

The screenshot shows a Windows form titled 'PERSONAS DE UNA ESCUELA'. It contains a 'Tipo de persona' section with two radio buttons: 'Profesor' (selected) and 'Estudiante'. Below this is a 'Datos de la persona' section with three text boxes: 'Nombre', 'Sueldo', and 'Calificación'. To the right of these text boxes are two buttons: 'Capturar datos' and 'Mostrar datos'. Labels with leader lines point to various components: 'grbTipoPersona' points to the 'Tipo de persona' section; 'radProfesor' and 'radEstudiante' point to the radio buttons; 'grbDatos' points to the 'Datos de la persona' section; 'btnCapturarDatos' points to the 'Capturar datos' button; 'btnMostrarDatos' points to the 'Mostrar datos' button; 'txtNombre', 'txtSueldo', and 'txtCalificacion' point to their respective text boxes.

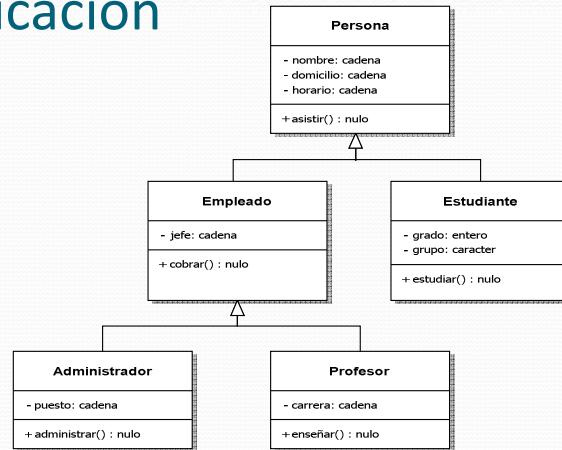
23

Salida del Prog. 5.2.- HerenciaFormas

The first screenshot shows a dialog box titled 'DATOS DEL PROFESOR'. It contains an information icon and the following text: 'Nombre: Bruno', 'Sueldo: \$1,234.00', 'Bruno asiste a la escuela ...', and 'Bruno está enseñando ...'. There is an 'Aceptar' button at the bottom. The second screenshot shows a dialog box titled 'DATOS DEL ESTUDIANTE'. It contains an information icon and the following text: 'Nombre: Paola', 'Calificación: 98', 'Paola asiste a la escuela ...', and 'Paola está estudiando ...'. There is an 'Aceptar' button at the bottom.

24

La herencia como estrategia de clasificación



Una clase base hereda todos sus componentes no privados y la clase derivada no puede elegir qué elementos recibir

25

Invocando un método de la clase base

- Una subclase puede llamar los métodos de su superclase con la palabra reservada “base”.

Se usa la palabra reservada “base” para invocar un método de una clase base desde una clase derivada, por lo tanto esta palabra NO puede usarse como el nombre de una variable

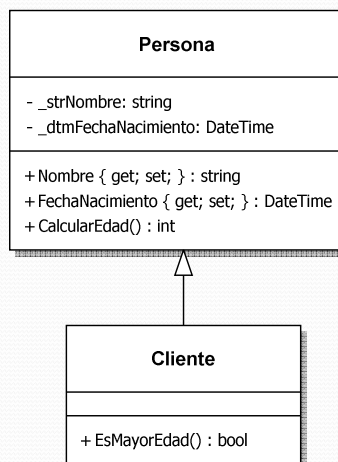
26

Ejercicio para invocar un método de una clase base desde una clase derivada

- Se desea determinar si un cliente es mayor de edad tomando como referencia su fecha de nacimiento. Para ello, se diseña un modelo orientado a objetos de una clase base **Persona** que define los datos nombre y la fecha de nacimiento de un individuo (con sus respectivas propiedades) y un método para determinar su edad (**CalcularEdad()**), que son heredados a una clase derivada identificada como **Cliente**.

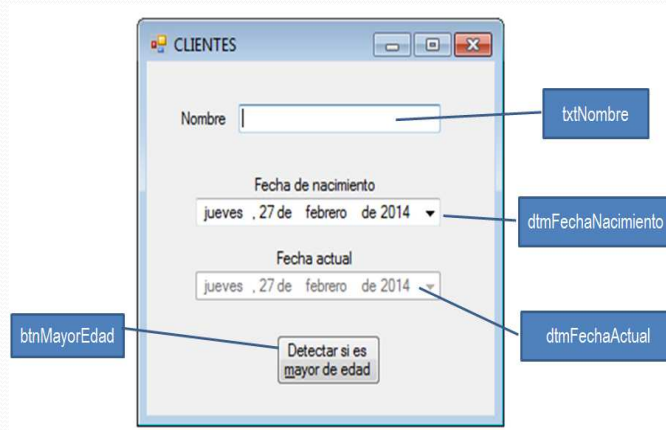
27

Prog. 5.3.- InvocandoMetodoClaseBase



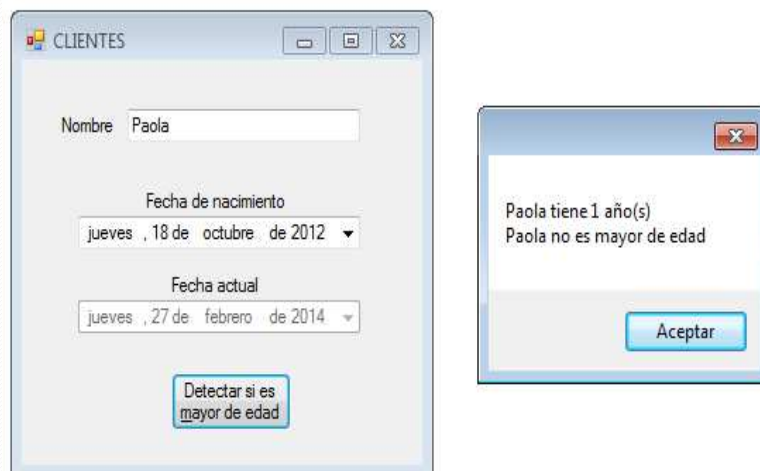
28

Prog. 5.3.- InvocandoMetodoClaseBase (cont.)



29

Prog. 5.3.- InvocandoMetodoClaseBase (cont.)



30

Codificación de la clase base

```
class Persona
{
    // Atributos privados
    private string _strNombre;
    private DateTime _dtmFechaNacimiento;

    // Propiedades públicas
    public string Nombre
    {
        get { return _strNombre; }
        set { _strNombre = value; }
    }

    public DateTime FechaNacimiento
    {
        get { return _dtmFechaNacimiento; }
        set { _dtmFechaNacimiento = value; }
    }

    // Método público para calcular la edad
    public int CalcularEdad()
    {
        int intEdad;
        TimeSpan intervalo;
        intervalo = DateTime.Now - this.FechaNacimiento;
        intEdad = (int)(intervalo.Days / 365.25);
        return (intEdad);
    }
}
```

31

Codificación de la clase derivada

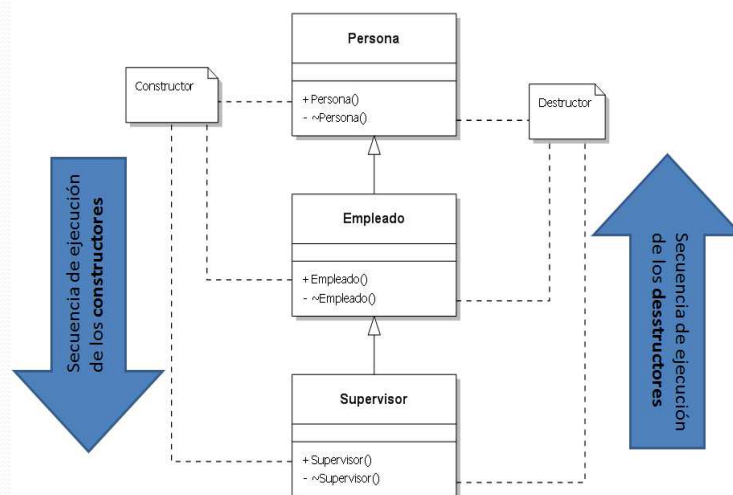
```
// La clase Cliente hereda de la clase Persona
class Cliente:Persona // Un cliente "es una" persona
{
    // Método público para determinar si es mayor de edad
    public bool EsMayorEdad()
    {
        // Variable local
        int intEdad;

        // Invoca el método CalcularEdad() de la clase base
        intEdad = base.CalcularEdad();

        if (intEdad >= 18)
            return (true);
        else
            return (false);
    }
}
```

32

Secuencia de ejecución de los constructores y destructores en la herencia



33

Invocando los constructores de la clase base

- También se puede invocar un constructor de la clase base desde el constructor de la clase derivada.
- Basta con definir el constructor de la clase derivada y colocar al final de su definición **:base(parámetros)**.
- Se puede invocar el constructor *default* (sin parámetros) o cualquier sobrecarga del constructor.

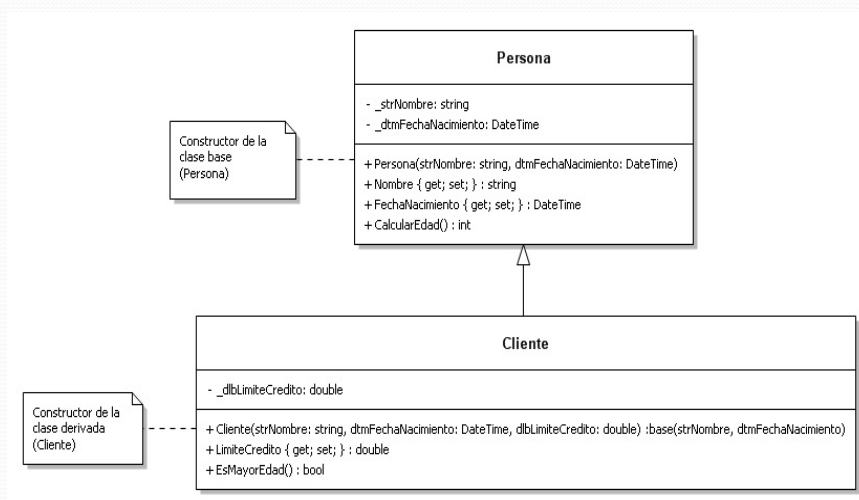
34

Ejercicio para invocar constructores de una clase base desde una clase derivada

- Una clase derivada llamada **Cliente** invoca el constructor de su clase base denominada **Persona**.

35

Prog. 5.4.- InvocarConstructorClaseBase



36

Prog. 5.4.-

InvocarConstructorClaseBase (cont.)

The screenshot shows a Windows form titled 'CLIENTES'. It contains several controls: a text box for 'Nombre', a text box for 'Límite crédito', two date pickers for 'Fecha de nacimiento' and 'Fecha actual', and a button labeled 'Detectar si es mayor de edad'. External labels with arrows point to these controls: 'txtNombre' points to the 'Nombre' text box, 'txtLimiteCredito' points to the 'Límite crédito' text box, 'dtmFechaNacimiento' points to the 'Fecha de nacimiento' date picker, 'dtmFechaActual' points to the 'Fecha actual' date picker, and 'btnMayorEdad' points to the 'Detectar si es mayor de edad' button.

37

Prog. 5.4.-

InvocarConstructorClaseBase (cont.)

The screenshot shows the 'CLIENTES' form with the following data entered: 'Nombre' is 'Pepe', 'Límite crédito' is '10000', 'Fecha de nacimiento' is 'martes .01 de junio de 1976', and 'Fecha actual' is 'jueves .27 de febrero de 2014'. The 'Detectar si es mayor de edad' button is highlighted. To the right, a message box is displayed with the text: 'Pepe tiene 37 año(s)', 'Pepe es mayor de edad', and 'Límite de crédito: \$10,000.00'. There is an 'Aceptar' button at the bottom of the message box.

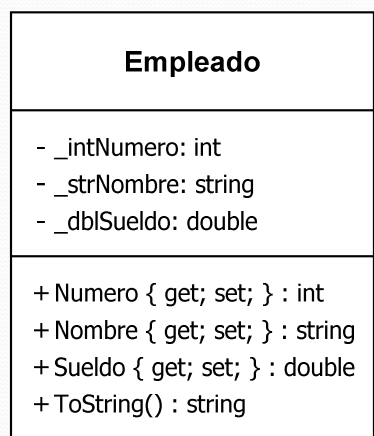
38

Sobrescritura del método ToString()

- El método `ToString()` está incluido en el framework .NET y se utiliza para convertir un dato a su representación de cadena (`string`).
- Todas las clases automáticamente heredan de la clase `System.Object`.
- Por lo tanto, el método `ToString()` puede ser sobrescrito (`override`) para ampliar su comportamiento y definir nuevas formas de desplegar datos.

39

Ejemplo



Se sobrescribe el método `ToString()`

40

Implementación

```
class Empleado
{
    . . .
    . . .
    // Sobrescribir el método ToString()
    public override string ToString()
    {
        return ("Datos del empleado:\n\nNúmero: " + this.Numero + "\nNombre: "
+ this.Nombre + "\nSueldo: " + this.Sueldo.ToString("C"));
    }
}
```

41

Salida

```
Empleado miEmpleado = new Empleado();
```

```
miEmpleado.Numero = 5;
```

```
miEmpleado.Nombre = "Bruno";
```

```
miEmpleado.Sueldo = 1234.56;
```

```
MessageBox.Show(miEmpleado);
```



En versiones de Microsoft Visual Studio anteriores a la 2015 es necesario codificar:

```
MessageBox.Show(miEmpleado.ToString());
```

42

Evitando la herencia: Clases selladas

- Las clases selladas (sealed) pueden ser instanciadas pero NO heredadas.
- Se utiliza la palabra “sealed” para indicarlo.
- Usar “sealed” simultáneamente con “abstract” produce un error.

ERROR!

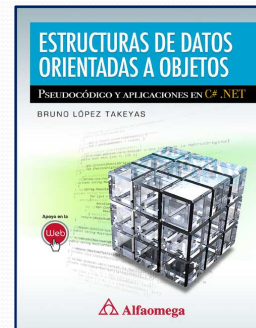
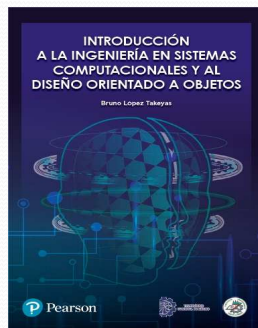
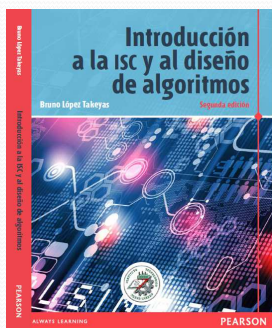
```
class Persona
{
    private string _strNombre;
    public string Nombre
    {
        get { return _strNombre; }
        set { _strNombre = value; }
    }
}

sealed class Empleado : Persona
{
    private string _strDepartamento;
    public string Departamento
    {
        get { return _strDepartamento; }
        set { _strDepartamento = value; }
    }
}

class EmpleadoTiempoParcial:Empleado
{
    private int _intHorasAsignadas;
    public int HorasAsignadas
    {
        get { return _intHorasAsignadas; }
        set { _intHorasAsignadas = value; }
    }
}
```

Otros títulos del autor

<http://www.itnuevolaredo.edu.mx/Takeyas/Libro>



✉ bruno.lt@nlaredo.tecnm.mx

 Bruno López Takeyas