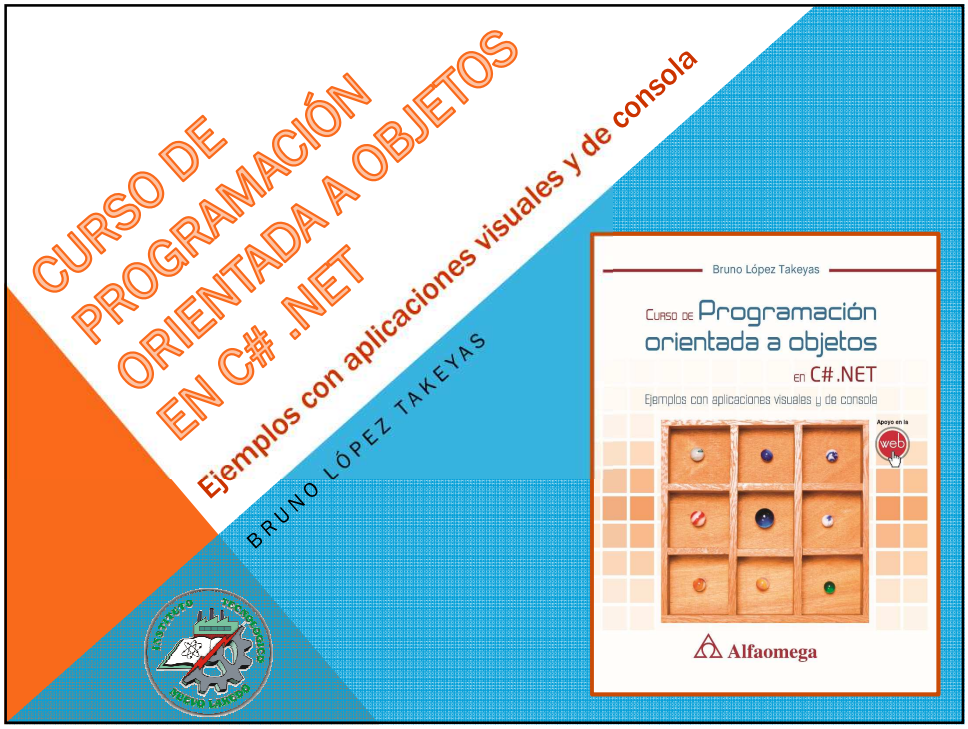

CURSO DE PROGRAMACIÓN ORIENTADA A OBJETOS EN C# .NET
Ejemplos con aplicaciones visuales y de consola
BRUNO LÓPEZ TAKEYAS
Bruno López Takeyas
Curso de Programación orientada a objetos en C#.NET
Ejemplos con aplicaciones visuales y de consola
Apoyo en la web
Alfaomega


CAPÍTULO 4
CLASES Y OBJETOS
2

Preguntas detonadoras

- ¿Cómo se pueden establecer restricciones de acceso a los componentes definidos en una clase?
- ¿Qué es un mutator y un accessor? ¿Para qué sirven?
- ¿A qué se refiere la sentencia this? ¿Para qué sirve? ¿Cuándo se utiliza?
- ¿Cuándo se recomienda utilizar una propiedad autoimplementada?
- ¿Se pueden definir varios métodos con el mismo nombre en una clase?
- El constructor de una clase, ¿crea un objeto?
- El destructor de una clase, ¿elimina un objeto?

3

Espacios de nombres (namespace)

- Organizan los diferentes componentes
- Un programa puede contener varios *namespaces*
- Un *namespace* puede contener muchas clases
- El programador puede crear sus propios *namespaces*
- Para accesar a *namespaces* se usa la directiva using:
  - `using System;`
  - `using System.Array;`

4

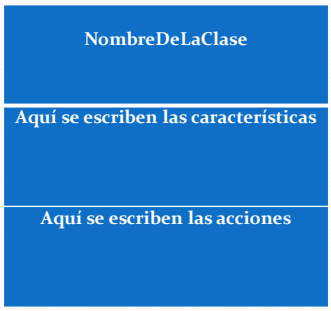
Clases y objetos

- Una clase es básicamente un plano para un tipo de datos personalizado.
- Cuando se define una clase, se utiliza cargándola en la memoria.
- Una clase que se ha cargado en la memoria se denomina *objeto* o *instancia*.
- Se crea una instancia de una clase utilizando la palabra clave de C# *new*

5

Clases en UML

- Cada clase se representa en un rectángulo con tres compartimentos:
  - Nombre
  - Atributos
  - Métodos y propiedades



6

Cómo declarar una clase

```
class nombre_de_la_clase
{
    … contenido de la clase …
}
```

7

Dentro de la clase…

- Se pueden declarar variables, propiedades, métodos, delegados, eventos, etc.
- Cada elemento puede tener un modificador de acceso.
- Un modificador de acceso especifica quienes están autorizados a "ver" ese elemento.
- Si no se especifica ningún modificador de acceso, se asume que se trata de un elemento "private".

8

Modificadores de acceso

- `public`
  Accesible a todos los elementos
- `private`
  Accesible solo a esa misma clase
- `protected`
  Accesible solo a la misma clase y métodos de sus clases derivadas. No accesible desde el exterior.
- `internal`
  Accesible solo a ese ensamblado
- `protected internal`
  Accesible desde el mismo ensamblado, la misma clase y métodos de sus clases derivadas

9

Modificadores de acceso

	Accesible desde ...				
Modificador de acceso	Clase donde se declaró	Subclase (Mismo assembly)	Subclase (Distinto Assembly)	Externamente (Mismo Assembly)	Externamente (Distinto Assembly)
private	SI	NO	NO	NO	NO
internal	SI	SI	NO	SI	NO
protected	SI	SI	SI	NO	NO
protected internal	SI	SI	SI	SI	NO
public	SI	SI	SI	SI	SI

10

Ejemplo: Variables con modificadores de acceso

```
class claseA
{
      //Si no se indica, es private
   int numero;
   private int numero1;

   public int numero2;
   protected int numero3 = 99;
   internal int numero4;
   protected internal int numero5;
}
```

11

Representación de modificadores de acceso en C# y UML

Modificador de acceso	Codificación en C#	Representación en UML
Privado	private	-
Público	public	+
Protegido	protected	#
Interno	internal	~
Protegido interno	protected internal	#

12

Miembros estáticos y de instancia

- *Miembro estático (static):* Sólo se crea una copia del miembro de la clase. Se crea cuando se carga la aplicación que contiene la clase y existe mientras se ejecute la aplicación
- *Miembro de instancia:* Se crea por default. Se crea una copia para cada instancia de la clase.

13

Miembros estáticos y de instancia

- Un miembro estático es un método o campo al que se puede obtener acceso sin hacer referencia a una instancia determinada de una clase
- No es necesario crear una instancia de la clase contenedora para llamar al miembro estático
- Cuando se tiene acceso a métodos estáticos, puede utilizar el nombre de clase, no el nombre de instancia

14

Declaración de miembros estáticos

- Cuando declara un campo de clase estático, todas las instancias de esa clase compartirán ese campo
- Una clase estática es una cuyos miembros son todos estáticos

```
static void Main(string[] args)
{
    Console.WriteLine("Tec Laredo");
}
```

Miembro estático

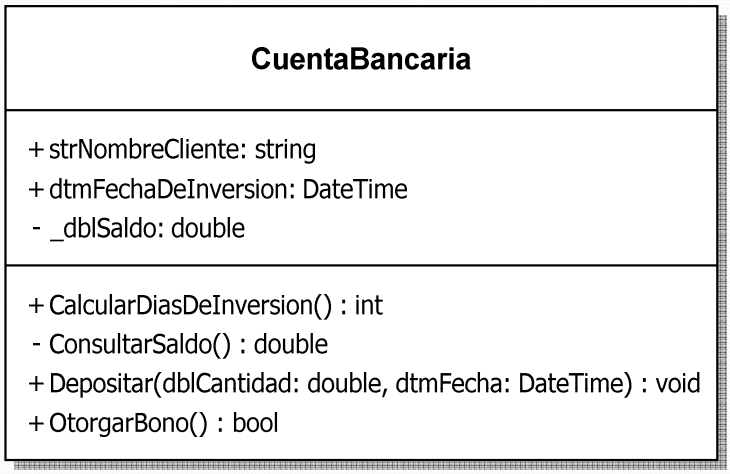
15

Atributos o campos

- Un atributo o campo es un dato común a todos los objetos de una determinada clase.
- Las variables declaradas dentro de una clase son ejemplos de atributos o campos

16

Ejemplo de una clase



17

Ejemplo de una clase

```
class CuentaBancaria
    {
        // Declaración de los atributos
        public string strNombreCliente;
        public DateTime dtmFechaDeInversion;
        private double _dblSaldo;

        // Defincón de los métodos
        public int CalcularDiasDeInversion()  {
            // Aquí se escribe el código de este método
        }

        private double ConsultarSaldo()  {
            // Aquí se escribe el código de este método
        }

        public void Depositar(double dblCantidad, DateTime dtmFecha)  {
            // Aquí se escribe el código de este método
        }

        public bool OtorgarBono()  {
            // Aquí se escribe el código de este método
        }
    }
```

18

Crear objetos: Instanciación

- Una vez creada la clase, ya es posible "consumirla" mediante la instanciación.
- La instanciación es el proceso de crear objetos a partir de una clase.
- Ejemplo en dos pasos:

```
ClaseA miobjetoA;
miobjetoA = new ClaseA();
```

1. Definir el tipo de "miObjetoA"
2. Adquirir memoria sin inicializar para el nuevo objeto usando new.
3. Ejecutar el constructor para inicializar la memoria y convertirla en un objeto usable

- Ejemplo en un paso:

```
ClaseA miobjetoA = new ClaseA();
```

19

Objetos en UML

- En UML un objeto se representa como un rectángulo con un nombre subrayado

Los objetos no se crean automáticamente cuando se declaran

20

Declaración de objetos globales

```
namespace CircunferenciaFormas
{
    public partial class Form1 : Form
    {

        // Aquí se declaran los objetos globales

       // Declaración del objeto global
       CuentaBancaria = cuentaCheques;
       // Creación del objeto global
       cuentaCheques = new CuentaBancaria();

       // Declaración y creación del objeto global en una sola línea
       CuentaBancaria cuentaAhorros = new CuentaBancaria();


        public Form1()
        {
            InitializeComponent();
        }

        ...
```

Accediendo a los miembros de los objetos

Asignación:

```
miobjetoA . Numero2 = 50;
```

Nombre del objeto — Selección — Miembro

Lectura:

```
miOtraVariable = miobjetoA . Numero2;
```

22

Métodos

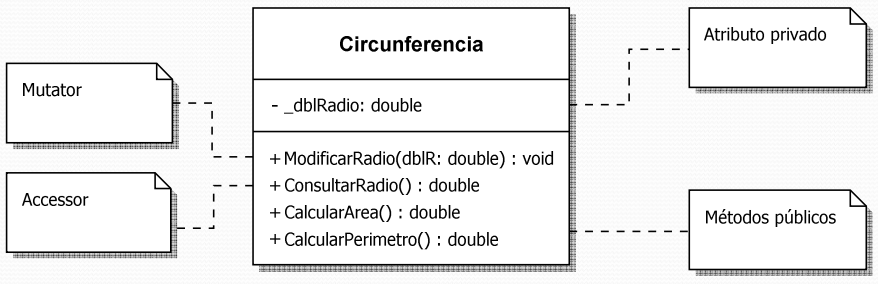
- Contienen instrucciones para ejecutar al momento de ser invocados.

- Un método contiene:
  - Modificador de Acceso (Determina su visibilidad)
  - Tipo de dato (Devuelto al finalizar su ejecución)
  - Identificador (Nombre con el cual se invoca)
  - Parámetros (Cero o mas variables que recibe el método)

No es recomendable diseñar clases cuyos métodos implementen código para capturar y/o mostrar los valores de sus atributos, ya que se pierde la reusabilidad de esa clase en aplicaciones con otras plataformas

23

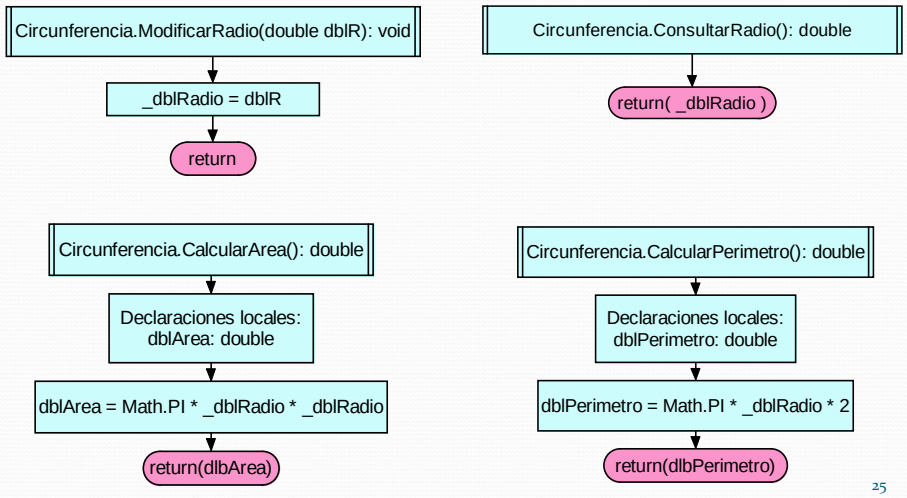
Uso de mutator y accessor



Al trabajar con objetos, primero deben introducirse los valores de sus atributos y después ejecutar las acciones invocando sus métodos.

24

Diagramas de flujo de los métodos de una clase



25

Codificación de la clase

```
class Circunferencia
    {
        // Declaración del atributo privado
        private double _dblRadio;

        // Mutator
        public void ModificarRadio(double dblR)
        {
            _dblRadio = dblR;
        }

        // Accessor
        public double ConsultarRadio()
        {
            return (_dblRadio);
        }

        // Método público para calcular el área
        public double CalcularArea()
        {
            // Declaración de variable local
            double dblArea;

            dblArea=Math.PI * _dblRadio * _dblRadio;

            return (dblArea); // Devuelve el resultado
        }

        // Método público para calcular el perímetro
        public double CalcularPerimetro()
        {
            // Declaración de variable local
            double dblPerimetro;

            dblPerimetro=Math.PI * _dblRadio * 2;

            return (dblPerimetro);  // Devuelve el resultado
        }
    }
```

26

La referencia *this*

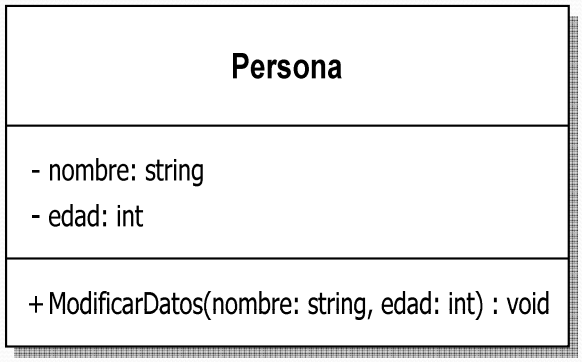
- Para hacer referencia (explícita) a un elemento que se encuentra dentro de la misma clase (ésta) se utiliza "this".

```
class Articulo
{   private double precio = 0;
    public void PonerPrecio ( double precio )
    {   this.precio = precio;
    }
    public double ConsultarPrecio()
    {   return this.precio;
    }
}
```

Muy útil cuando existen distintos elementos con el mismo nombre, pero con distinto significado dentro de un ámbito determinado.

27

Ejemplo de uso de la referencia *this*



El uso de la palabra this para referirse a los miembros internos de una clase es opcional, pero es necesaria cuando un parámetro y un atributo comparten el mismo nombre

28

Codificación de la referencia *this*

```
class Persona
    {
        // Declaración de los atributos privados
        private string nombre;
        private int edad;

        // Mutator
        public void ModificarDatos(string nombre, int edad)
        {
            this.nombre = nombre;
            this.edad = edad;
        }
    }
```

29

Propiedades

- Son mecanismos para acceder a los valores (variables) que poseen los objetos.
- Algunos autores las consideran sinónimos de los datos, sin embargo no siempre lo son (solamente en el caso de las propiedades autoimplementadas).

- Se pueden implementar:
  - **Declarando la variable como public**
    - No sería posible especificarla como Solo-Lectura o Solo-Escritura
    - No se puede implementar código adicional asociado a la Lectura o Escritura de la variable.
  - **Declarando un método de acceso a la variable**
    - Empobrece la legibilidad del código. Puede crear confusión.
  - **Utilizando accesadores get{ } y set { } para las propiedades**
    - Recomendado en C#

30

Implementando propiedades con accesadores get, set

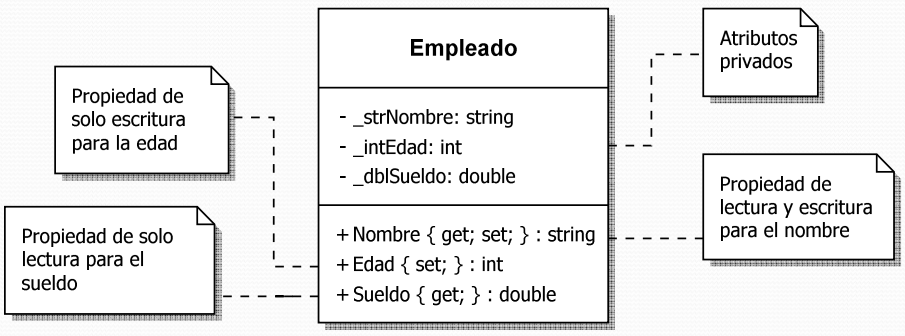
(Manera recomendada)

```
class Persona
{
    private int _intEdad;
    public int Edad
    {
        get
        {
            return _intEdad;
        }
        set
        {
            _intEdad = value;
        }
    }
}
```

```
class Programa
{
    static void Main()
    {
        Persona unaPersona = new
   Persona();
        unaPersona.Edad = 25;
        System.Console.WriteLine
                (unaPersona.Edad);
    }
}
```

31

Ejemplo de clase con propiedades



32

Codificación de la clase y sus propiedades

```
class Empleado
    {
        // Declaración de los atributos privados
        private string _strNombre;
        private int _intEdad;
        private double _dblSueldo;

        // Propiedad pública de lectura y escritura del nombre
        public string Nombre
        {
            get
            {
                return _strNombre;
            }
            set
            {
                _strNombre = value;
            }
        }

        // Propiedad pública de solo escritura para la edad
        public int Edad
        {
            set
            {
                _intEdad = value;
            }
        }

        // Propiedad pública de solo lectura para el sueldo
        public double Sueldo
        {
            get
            {
                return _dblSueldo;
            }
        }
    }
```

33

Validar datos mediante propiedades

```
// Propiedad pública de solo escritura para la edad

public int Edad
{
    set
    {
       if(value>=0)  // valida que la edad sea un número positivo
          _intEdad = value;  // modifica el valor del atributo
       else // si no
          _intEdad = 0; // arbitrariamente le asigna el valor cero
    }
}
```

34

Propiedades de solo lectura

- Si se desea que una propiedad sea de solo lectura, solo se debe incluir "get".

```
class Cuenta
{  private double _dblSaldo=0;
    public double Saldo
    {
        get { return _dblSaldo; }
    }
}
class Programa
{
    static void Main()
    {
        Cuenta miCuenta = new Cuenta();
        //miCuenta.Saldo = 32;
        System.Console.WriteLine(miCuenta.Saldo);
        System.Console.ReadLine();
    }
}
```

Realizar una asignación provocaría un error

35

Nivel de acceso asimétrico en las propiedades

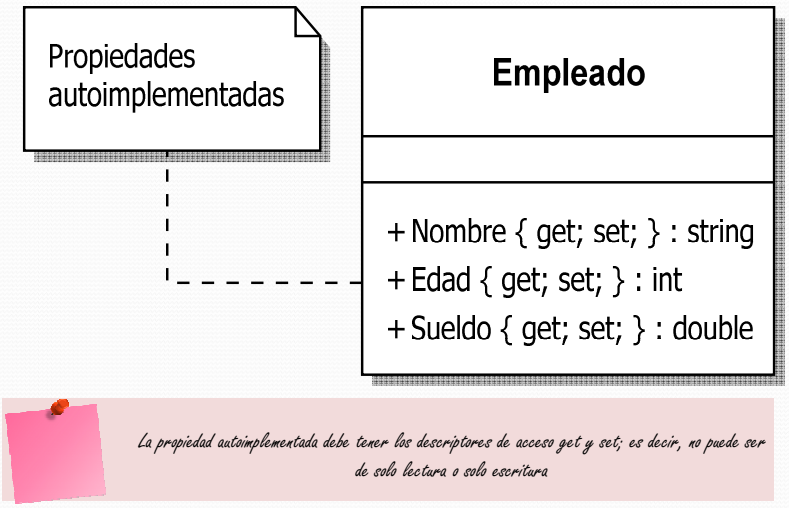
- Si una propiedad posee "get" y "set", entonces uno de los dos puede tener un modificador de acceso explícito.
- El nivel de acceso explícito de "get" o "set" debe ser mas restrictivo que el usado en la propiedad a la que pertenece.
- No se pueden utilizar en interfaces o implementación de ellas.

"get" sigue siendo public

```
class Empleado
{
    private int _intNumero;
    public int Numero
    {
                get
                {
                 return _intNumero;
                }
        private set
                {
                 _intNumero = value;
                }
    }
}
```

36

Propiedades autoimplementadas



La propiedad autoimplementada debe tener los descriptores de acceso get y set; es decir, no puede ser de solo lectura o solo escritura

37

Codificación de propiedades autoimplementadas

```
class Empleado
    {
        // Propiedades autoimplementadas
        public string Nombre
        {
            get;
            set;
        }

        public int Edad
        {
            get;
            set;
        }

        public double Sueldo
        {
            get;
            set;
        }
    }
```

38

Sobrecarga

- Se refiere a la posibilidad de crear métodos que posean el mismo nombre y difieran en la cantidad y/o tipo de parámetros (Firma).
  - No es posible crear 2 funciones con el mismo identificador que solo difieran en el tipo de dato devuelto.

```
static bool impresion(int num)
static char impresion(int num)
```

  - No es posible que 2 métodos sobrecargados solo difieran en el modificador "static".

39

Ejemplo de sobrecarga

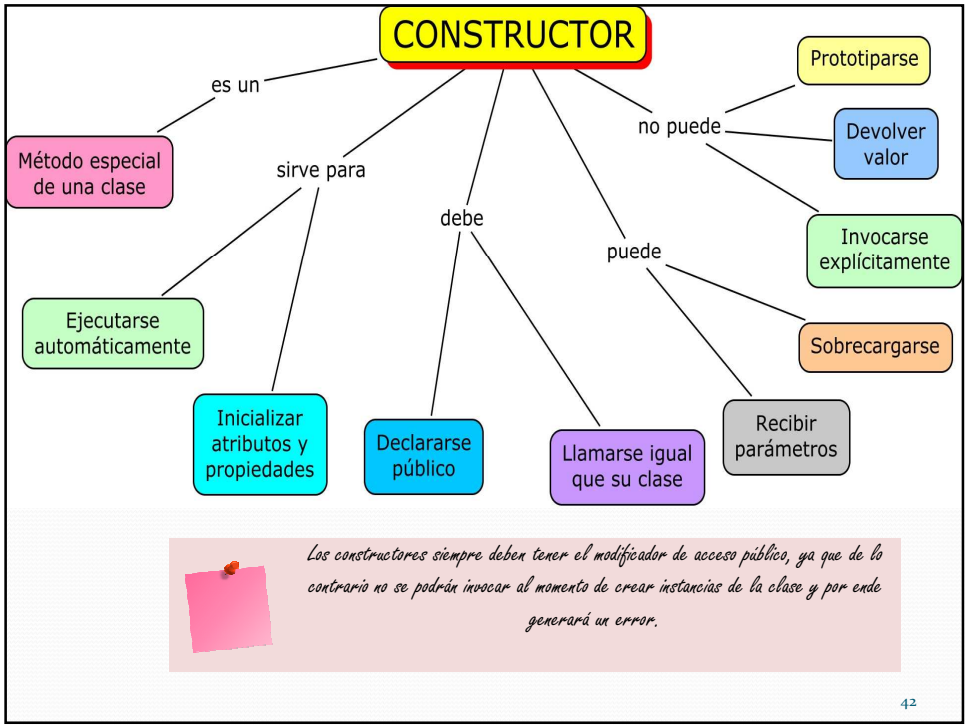
```
class Impresion
{   public void Imprimir()
    {   System.Console.WriteLine("Sistema de Usuarios");
    }
    public void Imprimir(string mensaje)
    {   System.Console.WriteLine(mensaje);
    }
    public void Imprimir (string titulo, string mensaje)
    {   System.Console.WriteLine("===== " + titulo + " =====");
        System.Console.WriteLine(mensaje);
    }
}
class Programa
{   static void Main()
    {
       Impresion im = new Impresion ();
       im.Imprimir();
       im.Imprimir("Siga las instrucciones en pantalla", "Use minusculas");
       im.Imprimir(" Gracias! ");
       System.Console.ReadLine();
    }
}
```

40

Método constructor

- Método especial que es invocado automáticamente cada vez que un objeto nuevo es creado (new).
- Debe poseer el mismo nombre de la clase.
- No posee tipo de dato de retorno.
- Si la clase no posee un constructor, C# crea uno vacío por default.

41



42

Ejemplo de constructor

```
class Persona
{
    private string _strNombre;

    public Persona()
    {
      _strNombre = "Desconocido";
    }
}
```

Constructor (notar que tiene el mismo nombre de la clase)

Dentro del constructor se realizan las inicializaciones correspondiente

```
class Programa
{
    static void Main()
    {
        Persona unaPersona = new Persona();
        System.Console.WriteLine(unaPersona.Nombre);
        System.Console.ReadLine();
    }
}
```

Ejemplo de constructor que recibe parámetros

```
class Persona
{   private string _strNombre;

    public Persona(string strNombre)
    {
        _strNombre = strNombre;
    }
}

class Programa
{   static void Main()
    {   Persona otraPersona = new Persona("Ramon");
        System.Console.WriteLine(otraPersona.Nombre);
        System.Console.ReadLine();
    }
}
```

Se debe proporcionar el valor de los parámetros al momento de la creación del objeto.

44

Ejemplo: Sobrecarga del constructor

```
class Persona
{   private string _strNombre;

    public Persona()
    {  _strNombre = "Desconocido";
    }
    public Persona(string strNombre)
    {  _strNombre = strNombre;
    }
}

class Programa
{   static void Main()
    {   Persona unaPersona = new Persona();
        System.Console.WriteLine(unaPersona.Nombre);

        Persona otraPersona = new Persona("Ramon");
        System.Console.WriteLine(otraPersona.Nombre);
        System.Console.ReadLine();
    }
}
```

Constructores

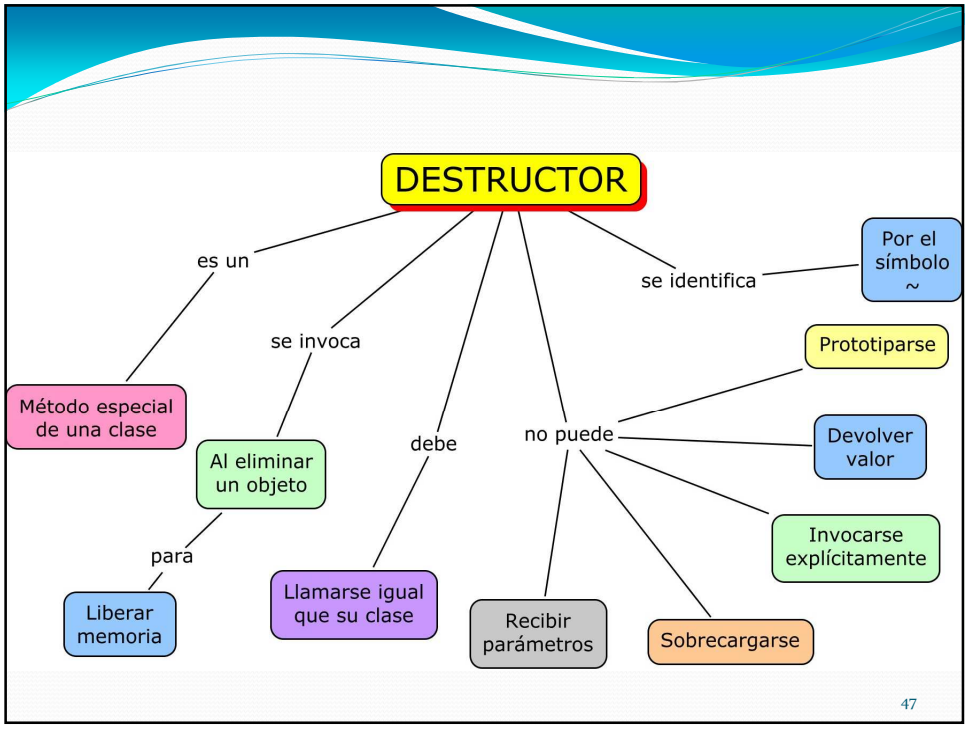
Dos constructores = Dos diferentes formas de crear objetos

45

Método destructor

- No puede llamarse explícitamente.
- Se invoca automáticamente cuando el objeto es destruido.
- Invocado por el Garbage Collector (GC) justo antes de desasignar la memoria del objeto.
- Usado para cerrar archivos, conexiones de red, canales abiertos, etc.
- Sintaxis similar al método constructor, pero anteponiendo una tilde (~) al nombre.

46



Ejemplo de destructor

```
class Empleado
{
    ~Empleado()
   {
    System.Console.WriteLine("Se ha destruido el Empleado ");
   }
}
```

Un destructor NO puede sobrecargarse, y se ejecuta cuando el recolector de basura destruye el objeto.

48

Destrucción de objetos

- C# gestiona la memoria de modo automático mediante el "recolector de basura" (Garbage Collector o GC), quien se encarga de eliminar la memoria que en algún momento se solicitó y no se ocupa mas.
- En C# no existe el operador contrario a "new".
- Por lo que, de manera general, NO es posible controlar exactamente el momento en el cual la memoria vuelve a estar disponible.

49

Destrucción de objetos

- Se puede indicar en el programa cuando un objeto deja de ser util, asignándole el valor "null".

```
miObjeto = null;
```

- Cuando el GC lo note, tomará cartas en el asunto.
- El instante preciso en que lo hará, queda fuera del alcance del programador.

50

El Garbage Collector (recolector de basura)

- **Se asegura que:**
  - Los objetos son destruidos una sola vez
  - Se destruyan objetos que ya no son utilizados.
- **Trabaja cuando:**
  - Hay poca memoria disponible
  - La aplicación está finalizando
  - El programador lo invoca manualmente (NO recomendado):

```
System.GC.Collect();
```

51

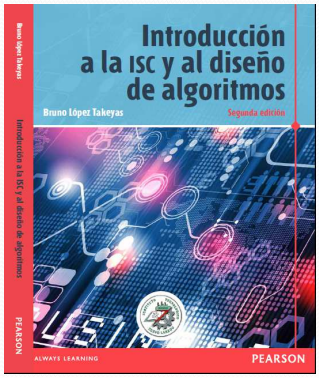
Consejos al diseñar clases y utilizar objetos

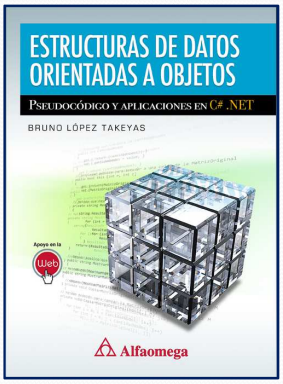
- No se deben capturar datos calculables
- Los datos calculables NO deben ser atributos
- Una vez creado el objeto, primero se deben introducir sus valores y después ejecutar sus métodos
- No se recomienda capturar ni imprimir datos desde un método ubicado dentro de una clase

52

Otros títulos del autor

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





takeyas@itnuevolaredo.edu.mx

Bruno López Takeyas