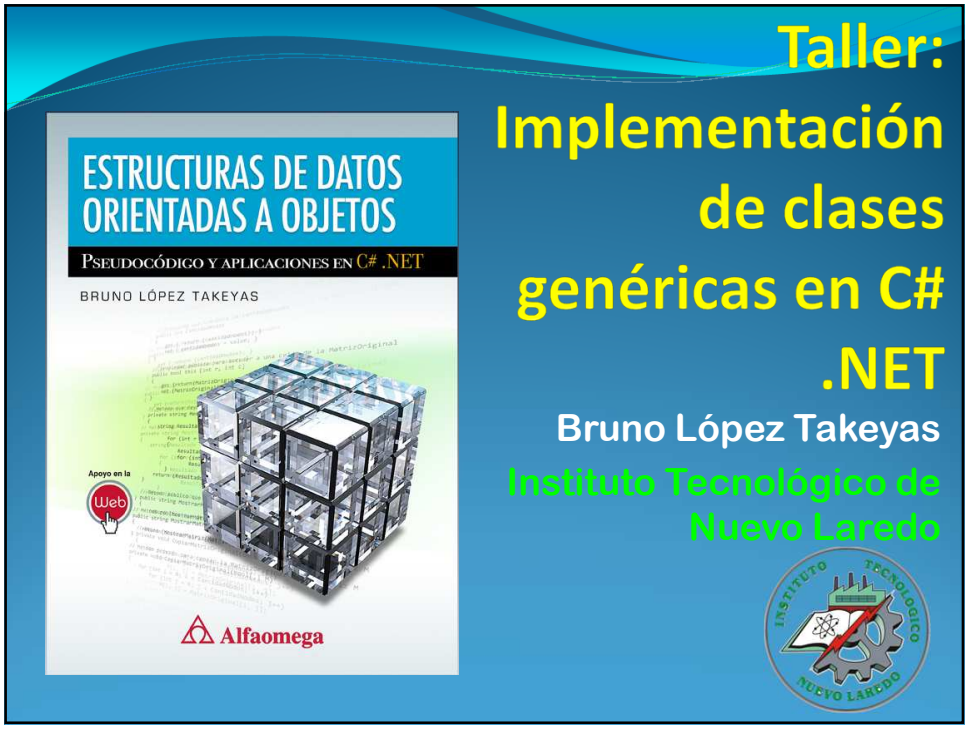
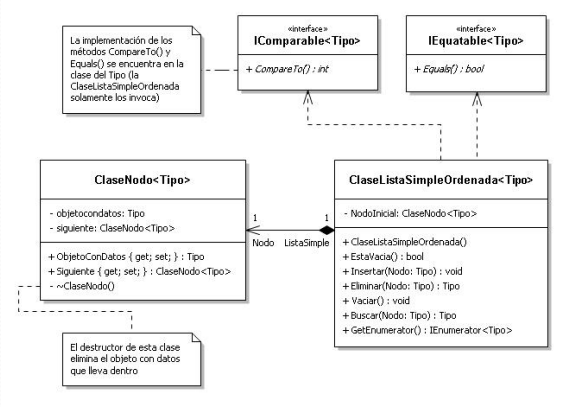


TALLER

1. Introducción a la relación de composición entre clases
2. Diseño del proyecto: Una empresa se compone de muchos empleados
3. Implementación en C# .NET
4. Para programadores avanzados …

1ª. PARTE

INTRODUCCIÓN A LA RELACIÓN DE COMPOSICIÓN ENTRE CLASES



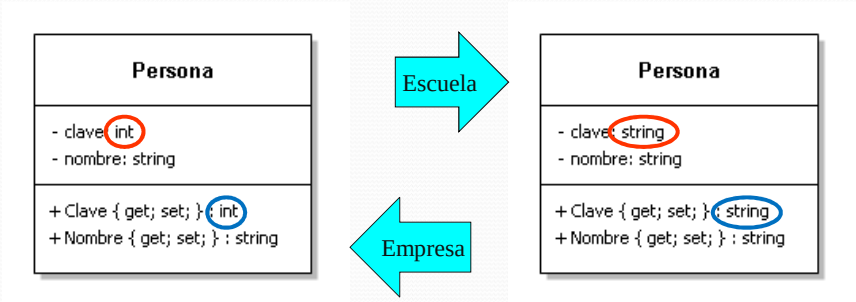
Clases parametrizadas o genéricas

- Ejemplo: Una empresa y una escuela desean almacenar la clave y nombre de sus personas:
  - *Clave: Entero ó String*
  - *Nombre: String*
- Pero en la empresa la clave es numérica entera y en la escuela es una cadena.

4

Clases parametrizadas o genéricas *(cont.)*

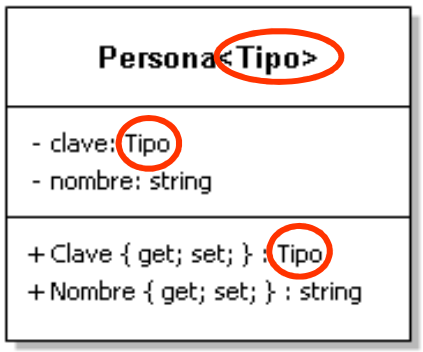
Clases iguales
(excepto en el tipo de dato de la clave)



5

Clases parametrizadas o genéricas *(cont.)*

Diseñar una clase parametrizada que sirva para ambos casos:



6

Colecciones genéricas en C#

- Incluidas en el namespace `System.Collection.Generic`
- Incorporadas a partir del .NET Framework 2.0
- Contiene clases e interfaces que definen tipos genéricos para instanciar colecciones
- Permite modelar estructuras de datos

7

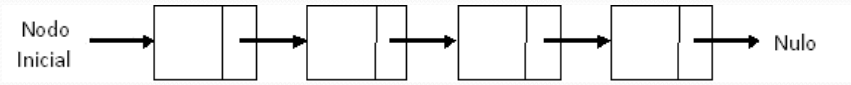
Estructuras de datos en C#

Colección (clase genérica)	Estructura de datos
`ArrayList`	Arreglos
`Stack`	Pilas
`Queue`	Colas
`List`	Listas enlazadas simples
`LinkedList`	Listas enlazadas dobles

8

Listas enlazadas simples

- Estructura de datos compuesta de nodos en secuencia enlazados a través de una referencia (apuntador).
- Cada nodo se compone de 2 partes:
  - Datos
  - Referencia al siguiente nodo
- Además, hay una referencia al primer nodo de la lista y el último nodo apunta a nulo



9

La clase genérica List

- Modela listas enlazadas simples en C#
- Requiere un parámetro adicional para definir el tipo de dato que almacena
- El parámetro se coloca entre < y >
- P. ejem.
  - `List <int> ListaSimpleEnteros;`
  - `List <double> ListaSimpleReales;`

10

Principales métodos y propiedades de la clase genérica List

Método o propiedad	Uso
`Clear`	Elimina todos los nodos de la lista
`Add`	Agrega un nodo al final de la lista
`Remove`	Elimina la primera ocurrencia de un nodo de la lista y devuelve un valor booleano para confirmar la operación
`Contains`	Determina si un nodo se encuentra almacenado en la lista
`Count`	Devuelve la cantidad de nodos almacenados en la lista
`Sort`	Ordena en forma ascendente los nodos de la lista

11

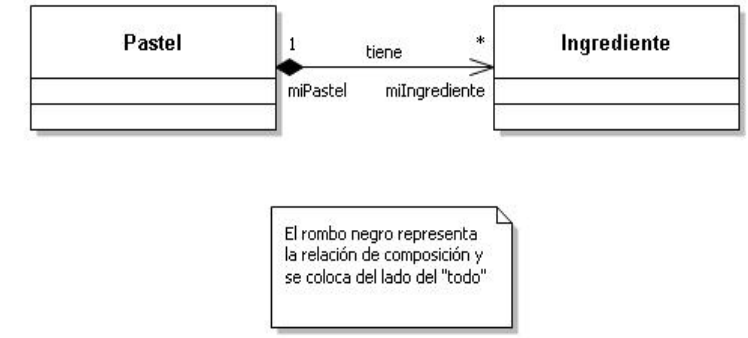
Composición

- Define una relación **fuerte** entre clases
- Se utiliza para modelar un **"todo"** y sus **"partes"** donde …
  - *El **"todo"** no puede existir si no existen sus **"partes"***
  - *Las **"partes"** desaparecen cuando se elimina el **"todo"***

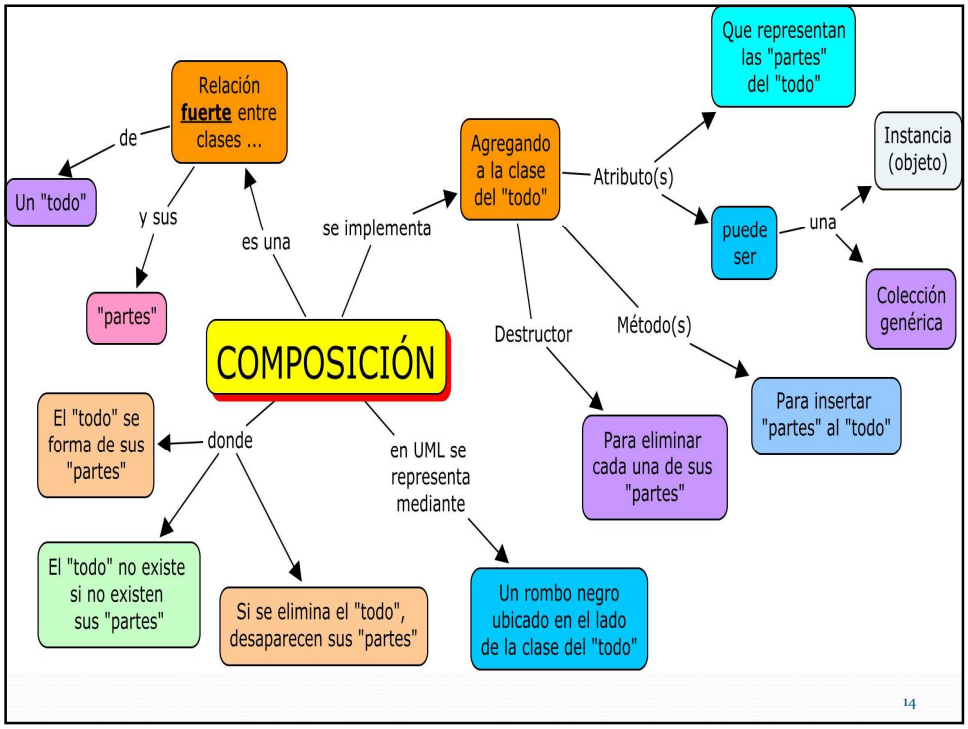
12

Representación de la composición

- Gráficamente se representa colocando un rombo **negro** en el extremo de la clase constituida (clase del “**todo**”).

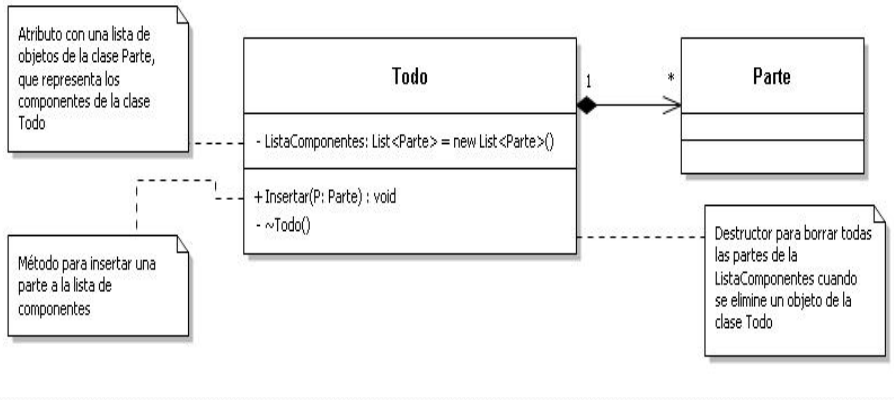


13



14

Relación de composición de 1..*



15

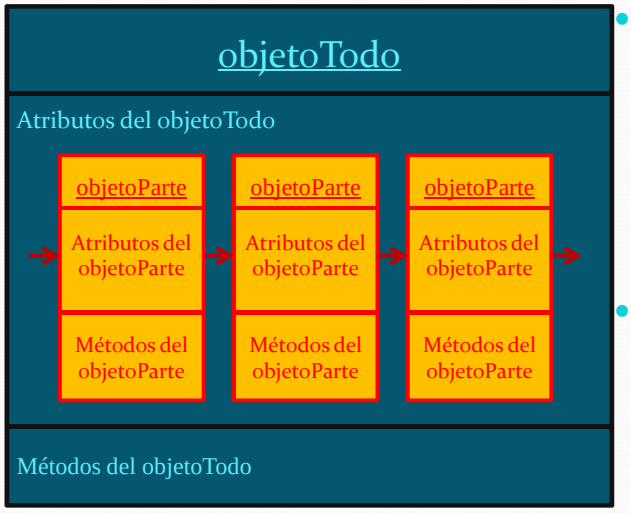
Implementación de relación 1..* en composición

```
class Todo
{
   // Atributo (lista de componentes del Todo)
   private List<Parte> ListaComponentes = new List<Parte>();

  // Método para insertar una parte a la lista
  public void Insertar(Parte P) {
      ListaComponentes.Add(P);
  }
  // Destructor (elimina el componente)
  ~Todo() {
      ListaComponentes.Clear();
  }
}
```

16

Representación de la relación 1..*

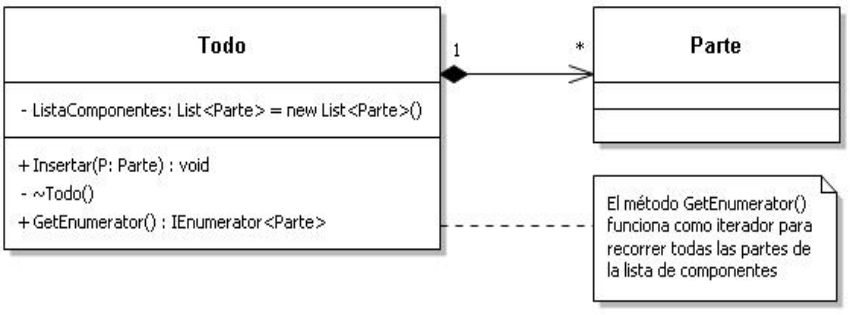


- Un objeto de la clase del “**todo**” tiene dentro una colección de objetos de la clase “**parte**”
- El **objetoTodo** tiene una lista de **objetoParte** como atributo

17

¿Cómo recorrer todas las partes de la ListaComponentes?

- Implementar un iterador por medio del método `GetEnumerator()`

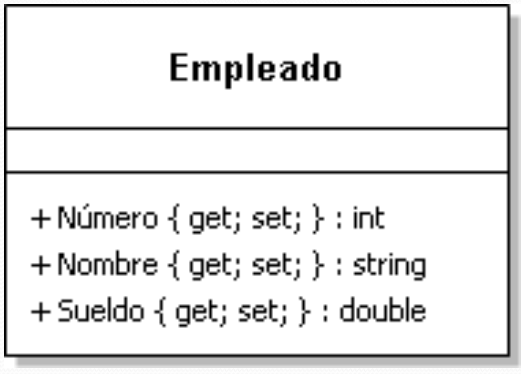


18

2ª. PARTE

DISEÑO DE LA APLICACIÓN: Una empresa se compone de muchos empleados

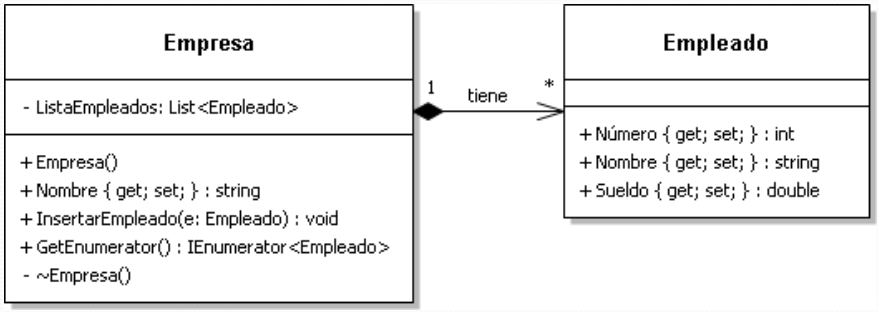
Datos de los empleados



- Crear una clase para modelar objetos con datos de empleados
- Crear una lista que almacene muchos empleados

20

Composición Empresa-Empleados

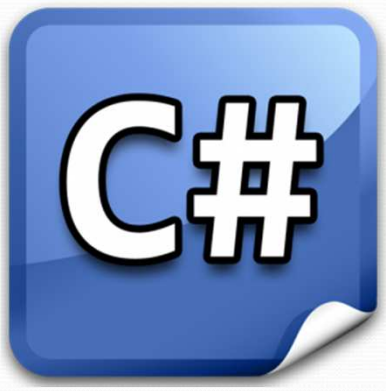


- Crear una empresa que tenga muchos empleados

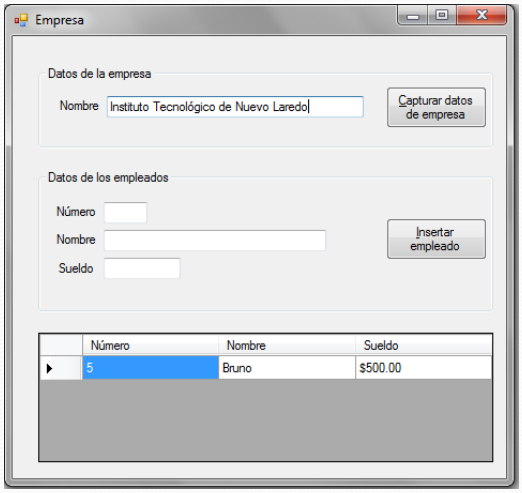
21

3ª. PARTE

IMPLEMENTACIÓN EN C# .NET



Diseño de la forma



- Capturar los datos de un empleado en *textBoxes*
- Al oprimir el botón Insertar empleado, crear un objeto de la clase `Empleado` e insertarlo en el objeto de la Empresa
- Mostrar los datos en un *dataGridView*

23

¿Cómo crear el dataGridView1?

```
private void Form1_Load(object sender, EventArgs e)
{
   dataGridView1.Columns.Add("Número", "Número");
   dataGridView1.Columns.Add("Nombre", "Nombre");
   dataGridView1.Columns.Add("Sueldo", "Sueldo");

   dataGridView1.Rows.Add(5);

   // Inicializa las propiedades del dataGridView1
   dataGridView1.ReadOnly = true;
   dataGridView1.AllowUserToAddRows = false;
   dataGridView1.AllowUserToDeleteRows = false;
   dataGridView1.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.Fill;

}
```

Codificación de la clase Empleado

```
class Empleado
    {
        // Propiedades públicas
        public int Numero
        {
            get;
            set;
        }

        public string Nombre
        {
            get;
            set;
        }

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

Codificación de la clase Empresa

```
class Empresa
{
  // Atributo con la lista de
Empleados
  private List<Empleado>
ListaEmpleados;

  // Constructor de la Empresa
  public Empresa()
  {
    ListaEmpleados = new
List<Empleado>();
  }

  // Propiedades públicas
  public string Nombre
  {
            get;
            set;
  }
```

```
 // Método para insertar un empleado
 public void InsertarEmpleado(Empleado
NuevoEmpleado)
 {
         ListaEmpleados.Add(NuevoEmpleado);
 }

 // Iterador para recorrer la
ListaEmpleados
 public IEnumerator<Empleado>
GetEnumerator()
 {
    return ListaEmpleados.GetEnumerator();
 }

 // Destructor que elimina la
ListaEmpleados al eliminar una Empresa
 ~Empresa()
 {
       ListaEmpleados.Clear();
 }
}
```

Creación de un objeto de una empresa

- Creación de un objeto de una empresa

```
Empresa miEmpresa = new Empresa();
```

- El objeto **miEmpresa** es un objeto global que contiene una lista genérica que almacena objetos de la clase **Empleado**

27

¿Dónde declarar y crear el objeto miEmpresa?

```
namespace Composicion_Empresa_Empleados
{
    public partial class Form1 : Form
    {
        // Declaración global de miEmpresa
        Empresa miEmpresa = new Empresa();

        public Form1()
        {
            InitializeComponent();
        }
      ……
```

¿Cómo insertar un empleado a mi empresa?

```
private void btnEmpleado_Click(object sender, EventArgs e)
{
  // Creación de un objeto para el NuevoEmpleado
  Empleado NuevoEmpleado = new Empleado();

  // Capturar datos del empleado en el NuevoEmpleado
  NuevoEmpleado.Numero = int.Parse(txtNumeroEmpleado.Text);
  NuevoEmpleado.Nombre = txtNombreEmpleado.Text;
  NuevoEmpleado.Sueldo = double.Parse(txtSueldo.Text);

  // Insertar el NuevoEmpleado a miEmpresa
  miEmpresa.InsertarEmpleado(NuevoEmpleado);

  MostrarDatosEmpleados();

  MessageBox.Show("Empleado insertado exitosamente");
}
```

¿Cómo mostrar los datos de empleados?

```
void MostrarDatosEmpleados()
{
  dataGridView1.Rows.Clear();  // Limpia el dataGridView1

  // Recorre la lista de empleados de miEmpresa por medio de su
iterador GetEnumerator() y los agrega al dataGridView1
  foreach (Empleado e in miEmpresa)
    dataGridView1.Rows.Add(e.Numero.ToString(), e.Nombre,
e.Sueldo.ToString("C"));

  // Limpia los textBoxes
  foreach (Control c in grbEmpleado.Controls)
    if (c is TextBox)
       c.Text = "";

  txtNumeroEmpleado.Focus();
}
```

4ª. PARTE

PARA PROGRAMADORES AVANZADOS …

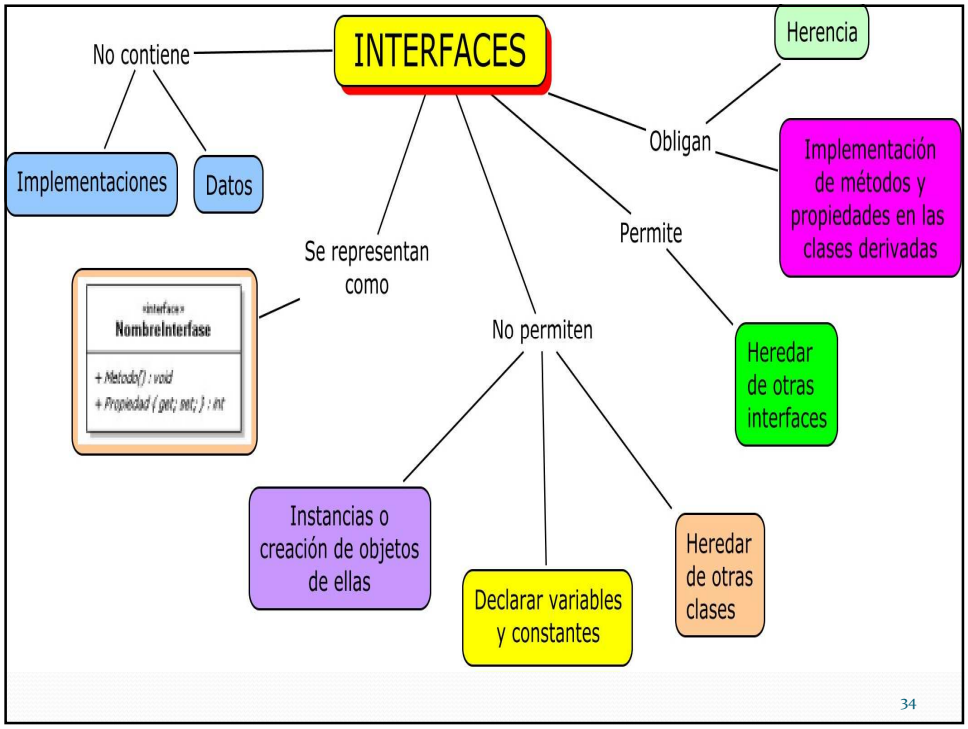
¿Se puede diseñar un sistema con varios propósitos?

- Supongamos que una empresa y una ferretería nos solicitan un sistema computacional para administrar los **empleados** y los **artículos** respectivamente.
- ¿Se puede hacer con el mismo diseño de proyecto?
- Pero …
  - Son datos diferentes
  - Diferentes variables
  - Diferentes tipos de datos
  - Criterios de ordenamiento distintos

Interfaces

- Son mecanismos para que puedan interactuar varios objetos no relacionados entre sí
- Son protocolos o “contratos” que obligan la herencia
- Contienen las declaraciones de los métodos, pero no su implementación.
- Son plantillas de comportamiento que deben ser implementados por otras clases.

33



34

Interfaces en C#

- IComparable
- IEquatable
- IEnumerable
- Y otras …

35

La interfaz IComparable

- Contiene la declaración del método `CompareTo()`

```
interface IComparable
{
   int CompareTo(object obj);
}
```

- El método `CompareTo()` devuelve un valor entero como resultado de la comparación

36

La interfaz IEquatable

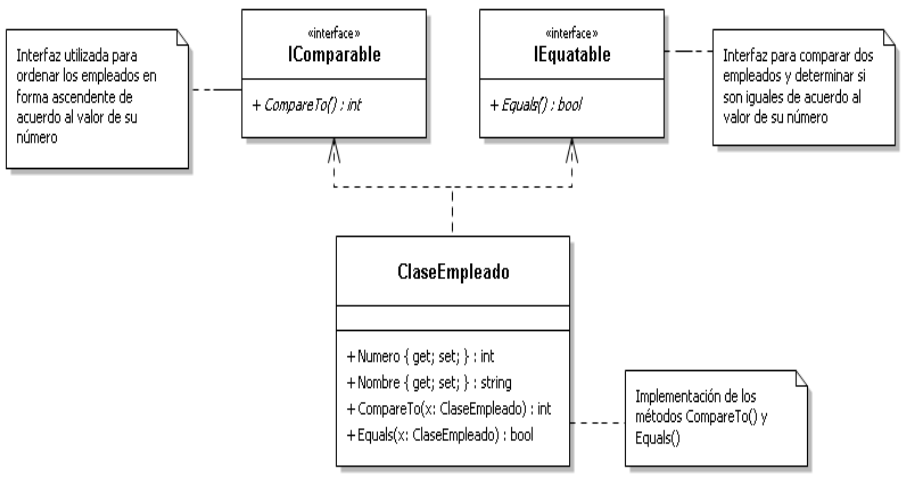
- Contiene la declaración del método `Equals()`

```
interface IEquatable<T>
{
        bool Equals(T obj);
}
```

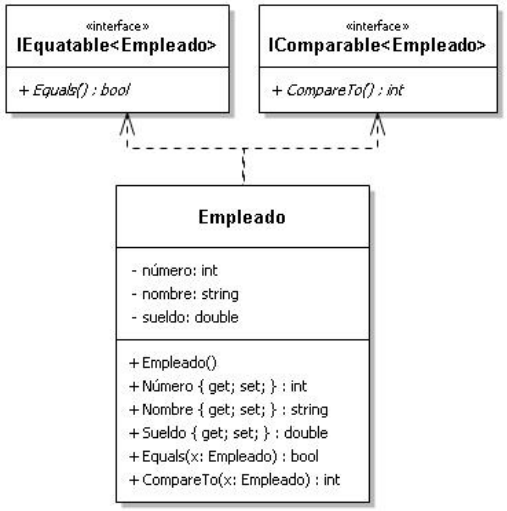
- El método `Equals()` devuelve un valor booleano como resultado de la comparación

37

Uso de las interfaces IComparable e IEquatable



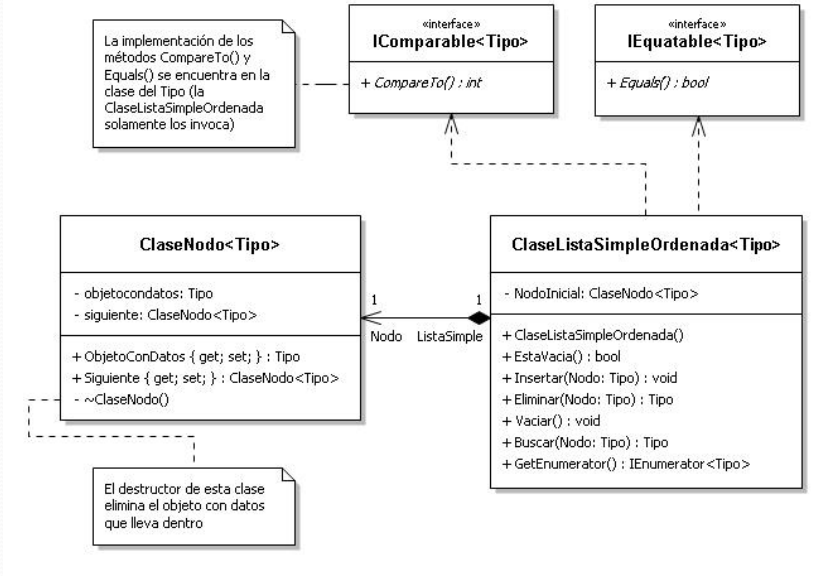
Datos de los empleados



- Crear una clase para modelar un objeto con datos de un empleado
- Crear una lista que almacene muchos empleados
- Ordenarlos descendentes de acuerdo al número

39

Diseño genérico de la lista



Creación de los objetos de las listas

```
ClaseListaSimpleOrdenada<Empleado>
        ListaEmpleados = new
ClaseListaSimpleOrdenada<Empleado>();


ClaseListaSimpleOrdenada<Artículo>
        ListaArtículos = new
ClaseListaSimpleOrdenada<Artículo>();
```

Diseño de clases genéricas

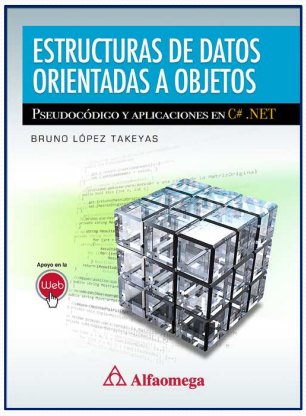
- **Objetos cuyos métodos y propiedades almacenan datos…**
  - **De cualquier tipo**
  - **Independientemente del nombre**
  - **Cualquier criterio de ordenamiento**
  - **Sin necesidad de modificar el código**
- **Mediante diseño e implementación de:**
  - **Clases**
  - **Interfaces**
  - **Delegados**
  - **Relaciones: herencia, composición, agregación, etc.**

Recomendaciones (temas)

- Clases
- Objetos
- Herencia
- Composición
- Agregación
- Clases abstractas
- Clases parametrizadas (genéricas)
- Interfaces (IEquatable, IComparable)
- Iteradores (GetEnumerator)
- Polimorfismo

Mayores informes y pedidos

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



takeyas@itnuevolaredo.edu.mx Bruno López Takeyas