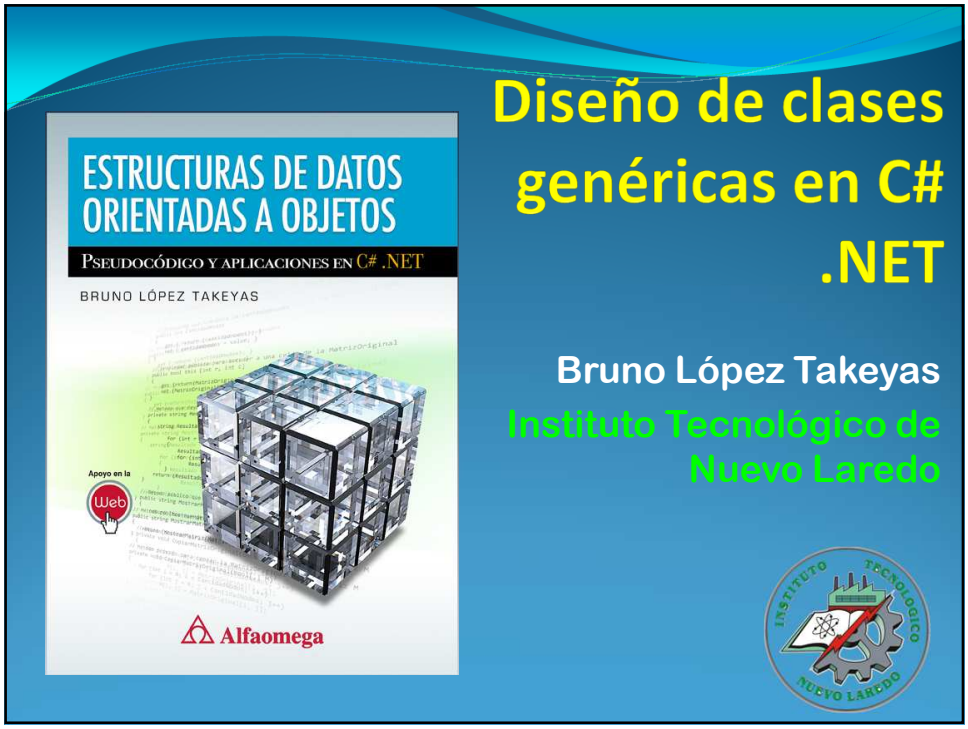


¿Se puede diseñar un sistema con varios propósitos?

- 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

CONFERENCIA

1. Introducción al paradigma orientado a objetos
2. Relaciones entre clases (herencia, composición y agregación)
3. Ejemplos de aplicación: Lista de empleados y lista de artículos

1ª. PARTE

INTRODUCCIÓN AL PARADIGMA ORIENTADO A OBJETOS

1.- Introducción al paradigma orientado a objetos

- La POO es un conjunto de técnicas que pueden utilizarse para desarrollar programas eficientemente.
- Los **objetos** son los elementos principales de construcción.
- La Orientación a Objetos (OO) es el estilo dominante de programación, descripción y modelado de hoy en dia.

Objetos en el mundo real

Lavadora

Perro

Televisión

Persona

Factura

6

Ejemplo: PERRO

- Características:
  - Nombre: “FIDO”
  - Raza: “Chihuahua”
  - Color: “Café”
  - ....etc...
- Acciones:
  - Ladrar [“Guau Guau”]
  - Comer [“Chomp Chomp”]
  - Dormir [“Zzzzzzzz”]
  - ...etc...

7

¿Cómo modelar un objeto real en un programa?

- Las “características” son ATRIBUTOS (datos) o PROPIEDADES.
- Las “acciones” son MÉTODOS u operaciones.

Objeto Perro “Real”

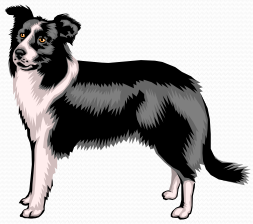
FIDO : Perro
Nombre: FIDO Raza: Chihuahua Color: Café
Ladrar() Comer() Dormir()

Abstracción de un objeto“Perro” en software

8

Clases y Objetos

- “FIDO” es UN “PERRO”
- “FIDO” es del TIPO “PERRO”
- “FIDO” es un **OBJETO**
- “PERRO” es la **CLASE** de “FIDO”



- “CHESTER” es OTRO “PERRO”
- “CHESTER” también es del TIPO “PERRO”
- “CHESTER” es otro **OBJETO**
- “PERRO” también es la **CLASE** de “CHESTER”

9

Clase

- Es una descripción de las características y acciones para un tipo de objetos.

- Una clase **NO** es un objeto. Es solo una plantilla, plano o definición para crear objetos.
- A partir de una clase se pueden crear muchos objetos independientes con las mismas características.

10

Clases en UML

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

11

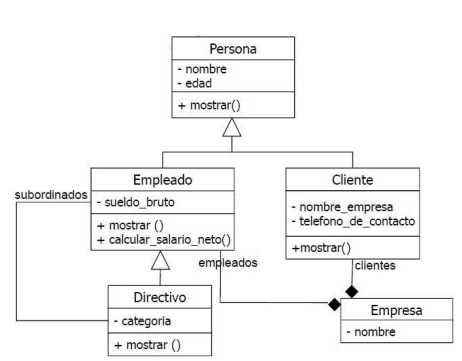
Objeto

- Unidad que combina datos y funciones.
  - Datos = Propiedades = Atributos = Características
  - Funciones = Métodos = Procedimientos =Acciones
- Un objeto es creado a partir de una clase.
- Los datos y funciones están **Encapsulados**.
- Posee un nombre único (identificador).
- Un objeto es del tipo de una clase
- “Un objeto es la instancia de una clase”
- Un objeto es un ejemplar específico creado con la estructura de una clase.

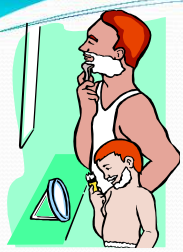
12

2ª. PARTE

RELACIONES ENTRE CLASES (Herencia, composición y agregación)



Herencia

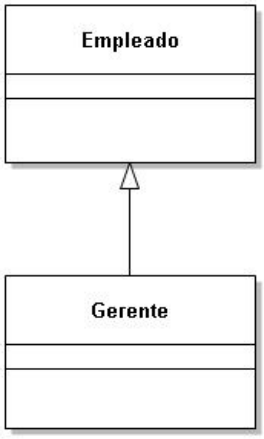


- Capacidad para utilizar características previstas en antepasados o ascendientes.
- Permite construir nuevas clases a partir de otras ya existentes, permitiendo que éstas les “transmitan” todos sus componentes.
- Una subclase hereda el comportamiento y la estructura de su Super Clase

14

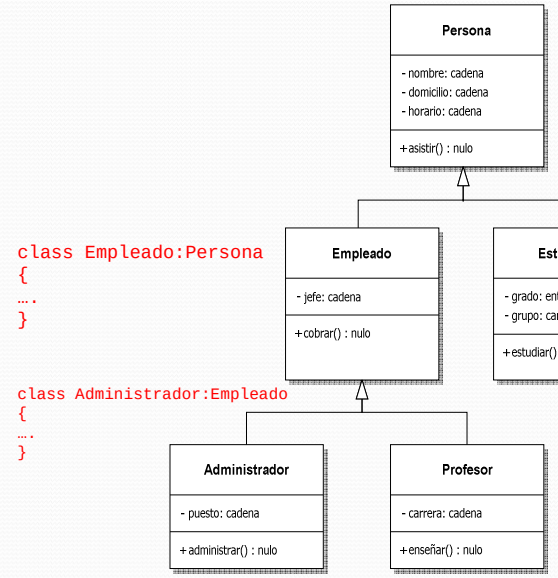
Representación de herencia en UML

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



15

Ejemplo de herencia




```
class Persona
{
….
}
```

```
class Empleado:Persona
{
….
}
```

```
class Estudiante:Persona
{
….
}
```

```
class Administrador:Empleado
{
….
}
```

```
class Profesor:Empleado
{
….
}
```

16

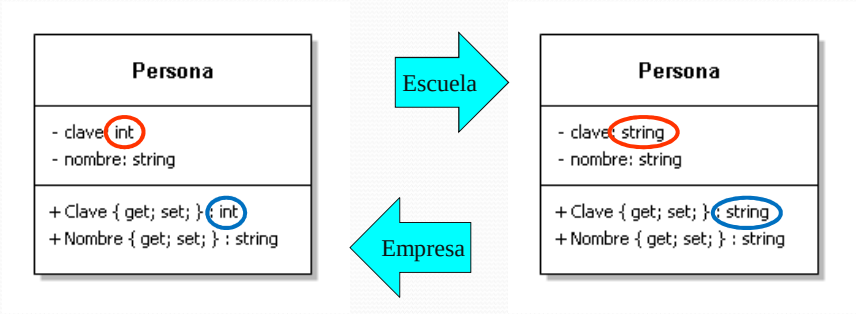
Clases parametrizadas o genéricas

- 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.

17

Clases parametrizadas o genéricas *(cont.)*

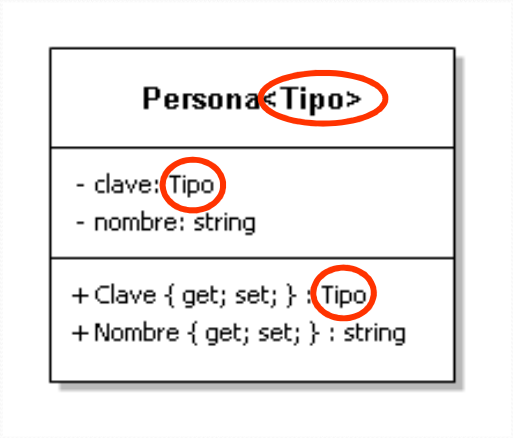
Clases iguales
(excepto en el tipo de dato de la clave)



18

Clases parametrizadas o genéricas *(cont.)*

Diseñar una clase parametrizada que sirva para ambos casos:



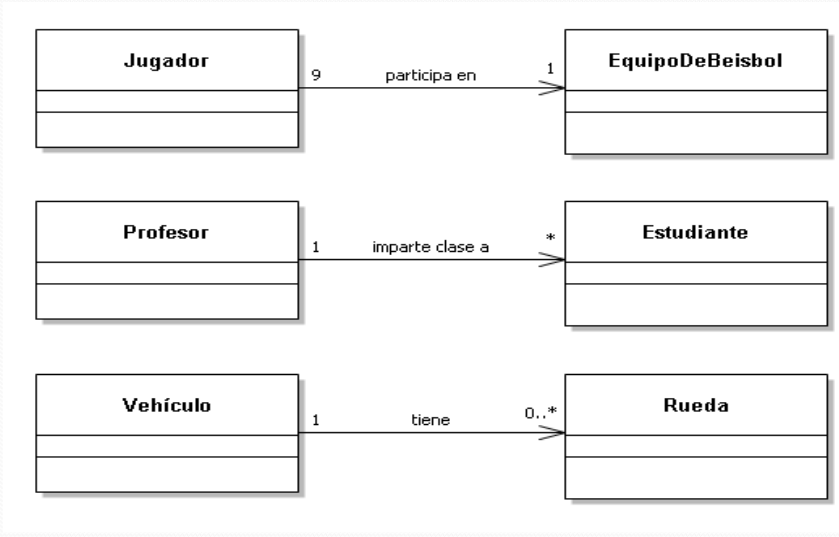
19

Asociación entre clases

- Asociación
  - Composición
  - Agregación

20

Ejemplos



21

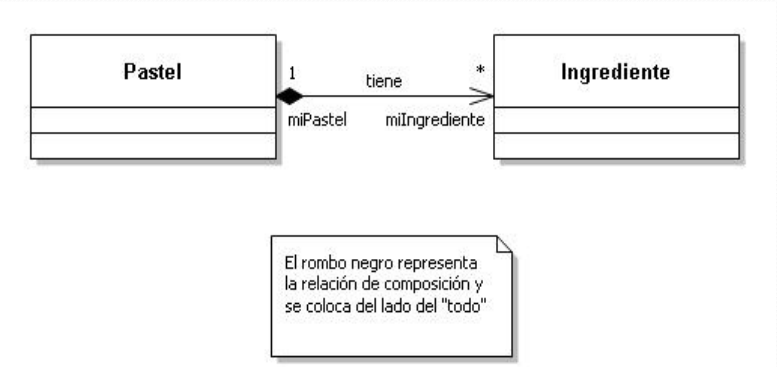
Composición

- 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"***

22

Representación de la composición

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



23

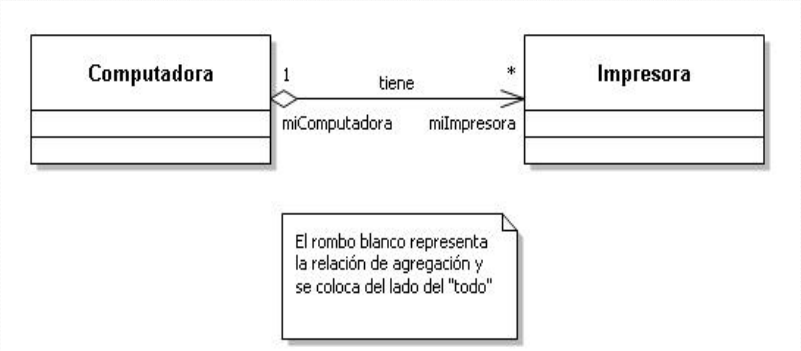
Agregación

- Define una relación **débil** donde una clase se puede formar de otras clases
- Sin embargo, la existencia de objetos de dichas clases es independiente
- Se utiliza para modelar un "**todo**" y sus "**partes**" donde …
  - *El "**todo**" se forma agregando sus "**partes**"*
  - *Las "**partes**" pueden pertenecer a varios "**todos**"*
  - *Si se elimina el "**todo**" pueden seguir existiendo sus "**partes**"*

24

Representación de la agregación

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

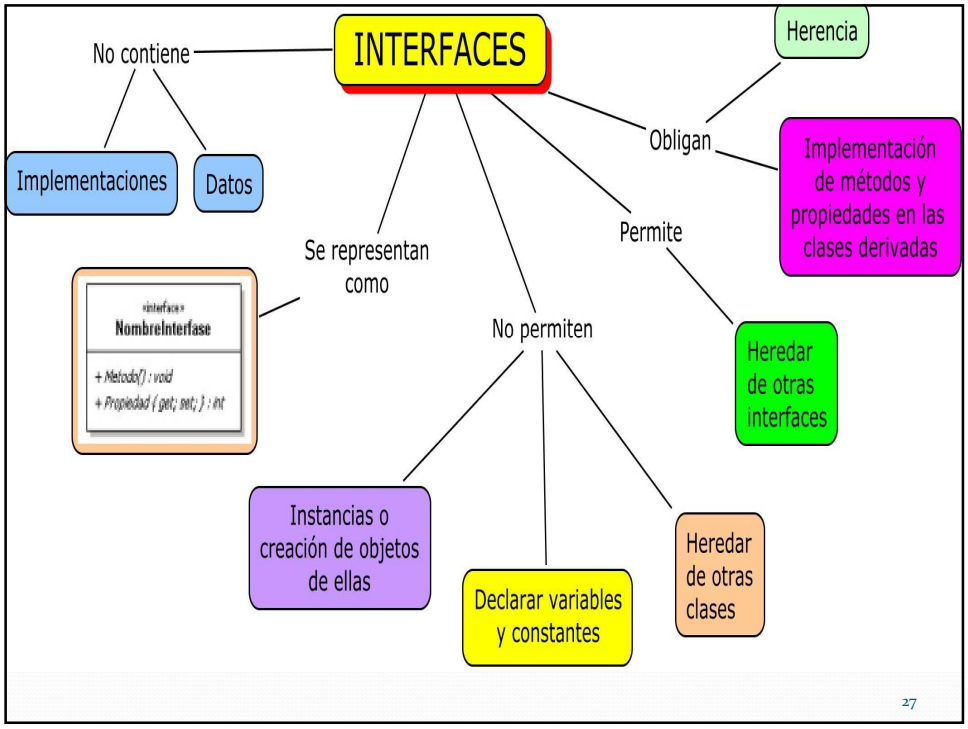


25

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.

26


INTERFACES
No contiene
Implementaciones
Datos
Herencia
Obligan
Implementación de métodos y propiedades en las clases derivadas
Permite
Heredar de otras interfaces
Se representan como
«interface»
NombreInterfase
+ Metodo() : void
+ Propiedad { get; set; } : int
No permiten
Instancias o creación de objetos de ellas
Declarar variables y constantes
Heredar de otras clases
27


Interfaces en C#
IComparable
IEquatable
IEnumerable
Y otras …
28

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

29

La interfaz IEquatable

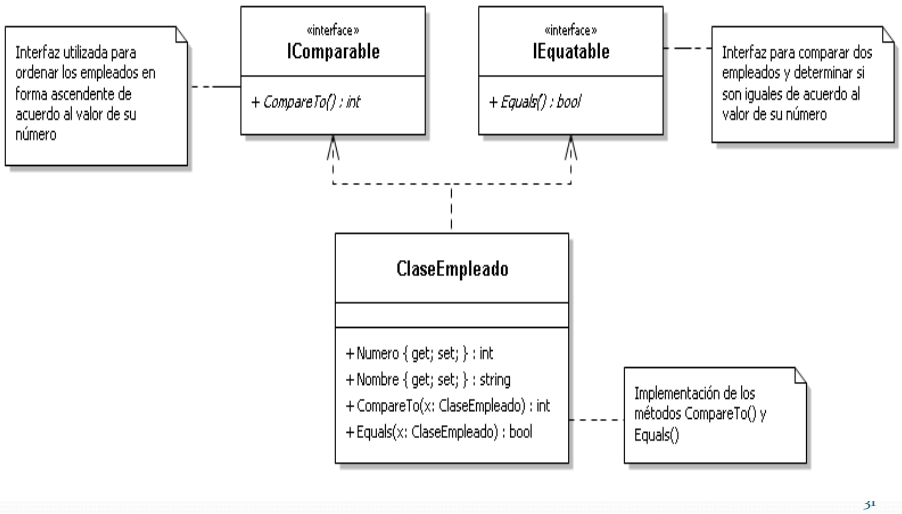
- Contiene la declaración del método `Equals()`

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

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

30

Uso de las interfaces IComparable e IEquatable

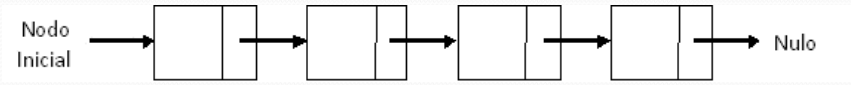


3ª. PARTE

EJEMPLOS DE APLICACIÓN: Lista de empleados y lista de artículos

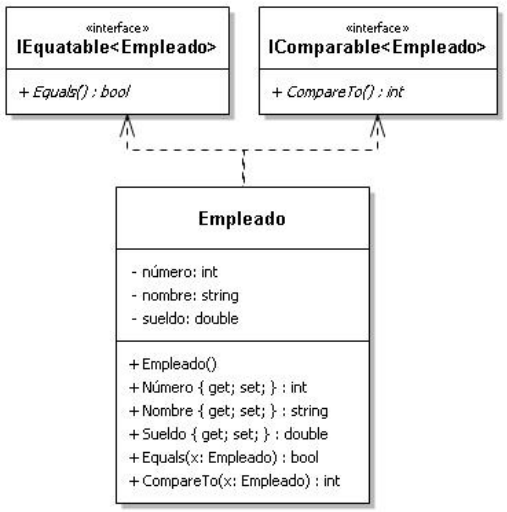
Listas enlazadas simples

- 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



33

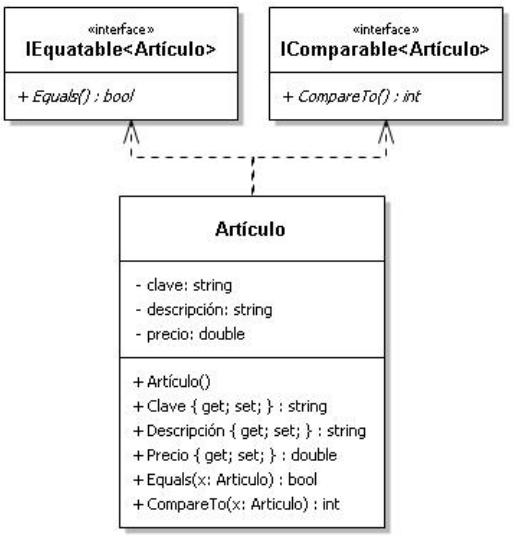
Datos de los empleados



- 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

34

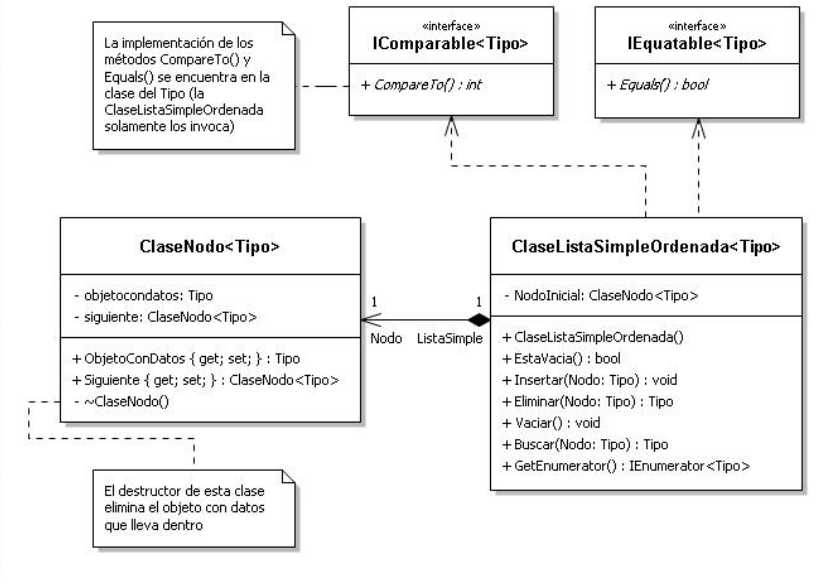
Datos de los artículos



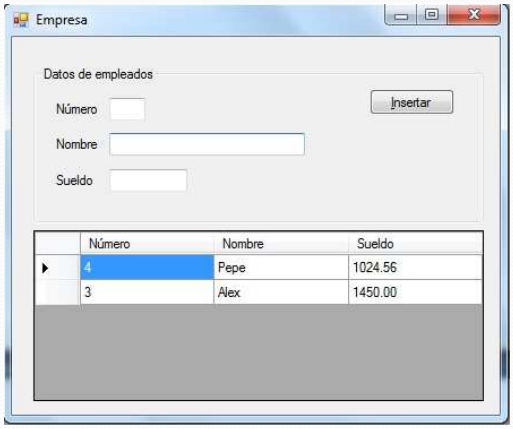
- Crear una clase para modelar un objeto con datos de un artículo
- Crear una lista que almacene muchos artículos
- Ordenarlos ascendentes de acuerdo a la clave

35

Diseño genérico de la lista



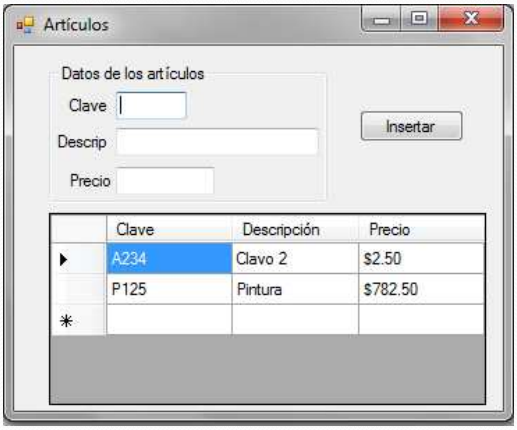
Insertar un empleado



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

37

Insertar un artículo



- Capturar los datos de un artículo en *textBoxes*
- Al oprimir el botón insertar, crear un objeto de la clase **`Artículo`** e insertarlo en la lista genérica
- Mostrar los datos en un *dataGridView*

38

Creación de los objetos de las listas

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


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

Diseño de clases genéricas

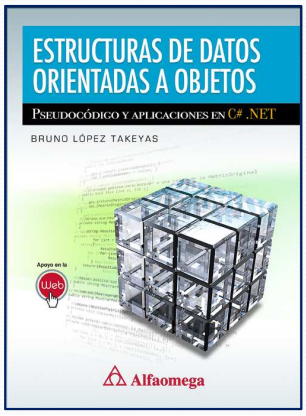
- **Objetos cuyos métodos y propiedades almacenan datos…**
  - **De cualquier tipo**
  - **Independientemente del nombre**
  - **Cualquier criterio de ordenamiento**
  - **Sin necesidad de modificar el código**
- **Mediante diseño e implementación de:**
  - **Clases**
  - **Interfaces**
  - **Delegados**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