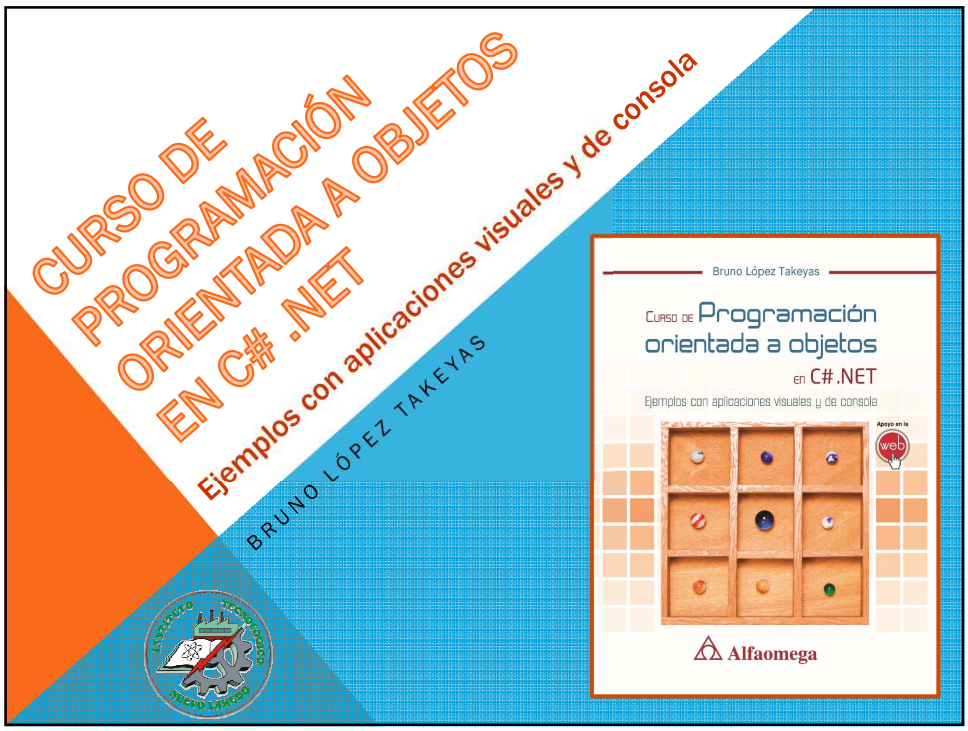
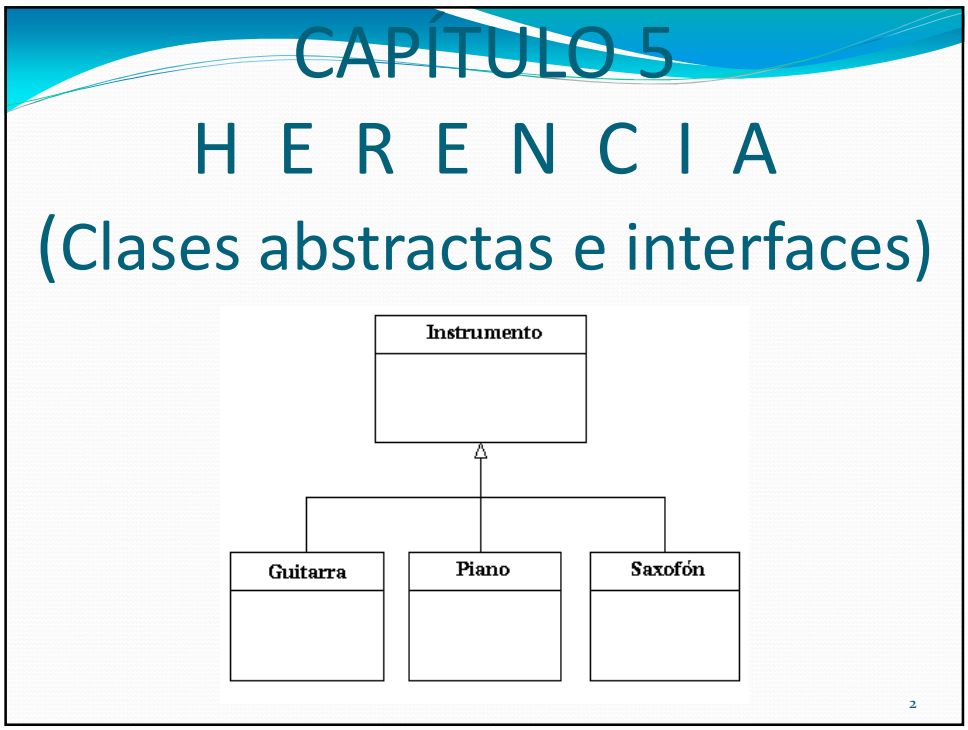

CURSO DE PROGRAMACIÓN ORIENTADA A OBJETOS EN C# .NET
Ejemplos con aplicaciones visuales y de consola
BRUNO LÓPEZ TAKEYAS
Bruno López Takeyas
Curso de Programación orientada a objetos en C#.NET
Ejemplos con aplicaciones visuales y de consola
Alfaomega


CAPÍTULO 5
H E R E N C I A
(Clases abstractas e interfaces)
Instrumento
Guitarra
Piano
Saxofón
2

Preguntas detonadoras

- Parece paradójico que una clase no pueda crear objetos a partir de ella, ¿realmente lo es?
- Si una clase abstracta no puede generar instancias, ¿entonces para qué sirve?
- Si un miembro abstracto no tiene implementación, ¿entonces para qué sirve?
- En una clase abstracta, ¿todos sus miembros son abstractos?
- ¿En qué se parece una interfase a una clase abstracta? ¿En qué difieren?
- ¿Se puede definir un miembro abstracto dentro de una clase no abstracta?

3

Clases abstractas e interfaces

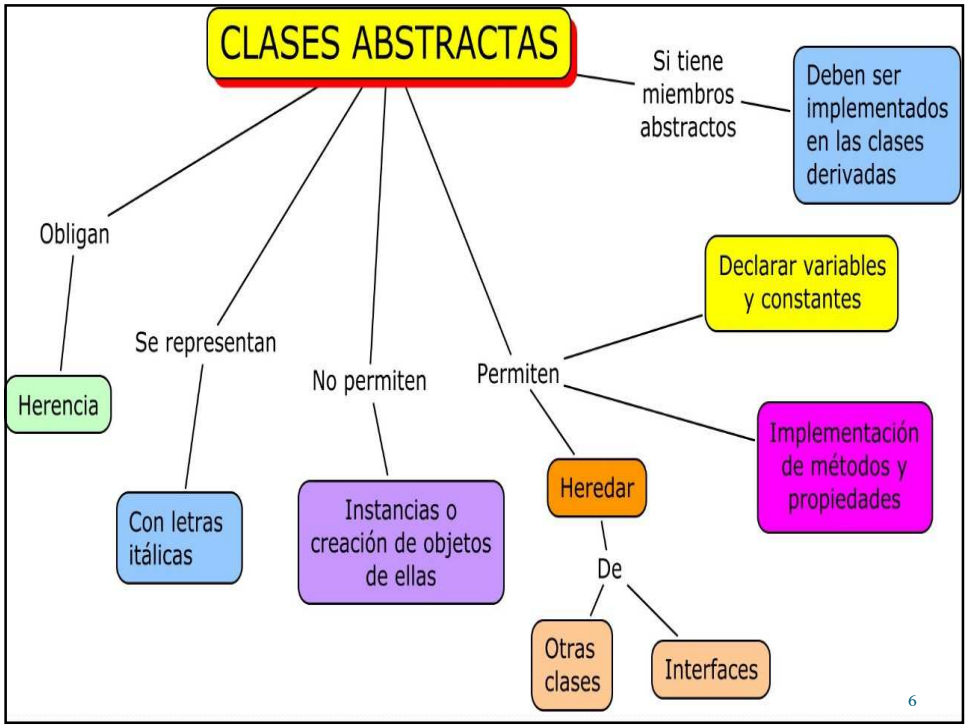
- Tanto las **clases abstractas** como las **interfaces** son mecanismos que obligan la herencia
- No se pueden instanciar, es decir, no se puede crear objetos de ellas

4

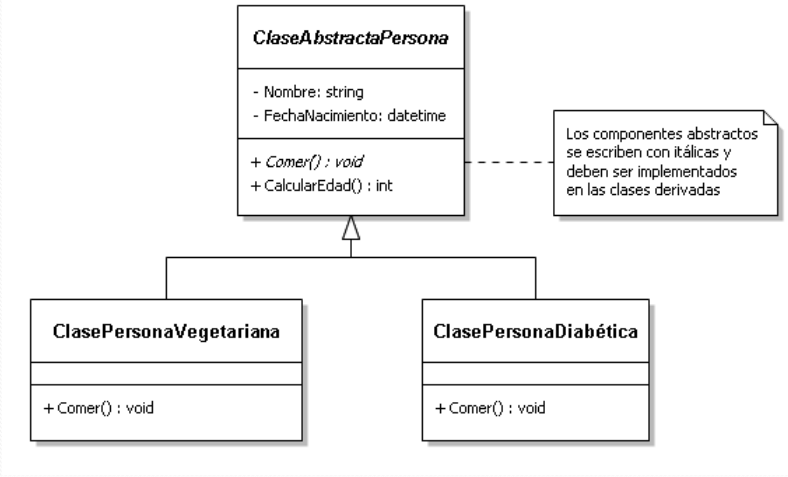
Clases abstractas

- Son mecanismos que obligan la herencia
- No se pueden instanciar, es decir, no se puede crear objetos de ellas
- Se utilizan solamente para heredar de ellas (Forzar u obligar la herencia).
- Se antepone la palabra "**`abstract`**" al nombre de la clase.

5



Ejemplo de clase abstracta



7

Implementación de una clase abstracta en C#

```
public abstract class ClaseAbstractaPersona
  {
      string Nombre;
      DateTime FechaNacimiento;

      public abstract void Comer();

      public int CalcularEdad()
      {
         //Aquí se implementa este método
      }
  }
```

Método abstracto (debe ser implementado en las clases derivadas)

8

Ejemplo: Clase Abstracta

```
abstract class Persona
{
    private string _strNombre;
    private string _strApellido;
    public string Nombre
    {
        get { return _strNombre; }
        set { _strNombre = value; }
    }
    public string Apellido
    {
        get { return _strApellido; }
        set { _strApellido = value; }
    }
    public string ObtenerNombreCompleto()
    {
      return
         this.Nombre + " " + this.Apellido;
    }
}
```

```
class Empleado : Persona
{
    private int _intClaveEmpleado;
    public int Clave
    {
     get { return _intClaveEmpleado; }
     set { _intClaveEmpleado = value; }
    }
}
class Cliente : Persona
{
    private string _strRfc;
    public string RFC
    {
        get { return _strRfc; }
        set { _strRfc = value; }
    }
}
```

Continuación… Ejemplo de Clase Abstracta

```
class Programa
{
    static void Main()
    {
        Empleado unEmpleado = new Empleado();
        unEmpleado.Nombre = "Juan";
        unEmpleado.Apellido = "Gonzalez";
        unEmpleado.Clave = 1;
        System.Console.WriteLine(unEmpleado.ObtenerNombreCompleto());

        Cliente unCliente = new Cliente();
        unCliente.Nombre = "Pedro";
        unCliente.Apellido = "Ramirez";
        unCliente.RFC = "RAHP780212";
        System.Console.WriteLine(unCliente.ObtenerNombreCompleto());

        System.Console.ReadLine();
    }
}
```

10

Clases abstractas con elementos abstractos

- Las clases abstractas *pueden* definir métodos y propiedades abstractos, con lo que su respectiva implementación en la subclase es obligatoria. (Los elementos abstractos DEBEN ser sobreescritos en la subclase).
- Se utiliza "`abstract`" para definir elementos abstractos (solo dentro de clases abstractas).
- Los elementos abstractos NO proporcionan implementación; solo declaraciones.
- En la subclase, se utiliza "`override`" para realizar la implementación correspondiente.

11

Miembros abstractos

- Una clase abstracta puede tener datos (atributos) e implementar métodos y propiedades como una clase normal y además puede tener miembros abstractos (métodos o propiedades).
- Los miembros abstractos NO tienen implementación (están vacíos).
- ¿Para qué sirve un método vacío o propiedad vacía y que no realiza acciones?

12

Ejemplo: Clase abstracta con elementos abstractos

```
abstract class Persona
{
    private string _strNombre;
    private string _strApellido;
    public string Nombre
    {
        get { return _strNombre; }
        set { _strNombre = value; }
    }
    public string Apellido
    {
        get { return _strApellido; }
        set { _strApellido = value; }
    }
    public abstract int Clave
    { get; set; }

    public abstract string ConsultarTodosLosDatos();

    public string ObtenerNombreCompleto()
    {
        return
        this.Nombre + " " + this.Apellido;
    }
}
```

Se DEBEN implementar estos elementos al heredar de esta clase.

13

Clase abstracta con elementos abstractos (cont.)

```
class Empleado : Persona
{
    private int _intClaveEmpleado;
    public override int Clave
    {
        get { return _intClaveEmpleado; }
        set { _intClaveEmpleado = value; }
    }
    public override string ConsultarTodosLosDatos()
    {
        return "------Datos del Empleado: \n" +  this.Clave + " " +
               this.Nombre + " " + this.Apellido;
    }
}

class Cliente : Persona
{
    private int _intClaveCliente;
    public override int Clave
    {
        get { return _intClaveCliente; }
        set { _intClaveCliente = value; }
    }
    public override string ConsultarTodosLosDatos()
    {
        return "******Datos del Cliente: \n" + this.Clave + " " +
                this.Nombre + " " + this.Apellido;
    }
}
```

Implementación

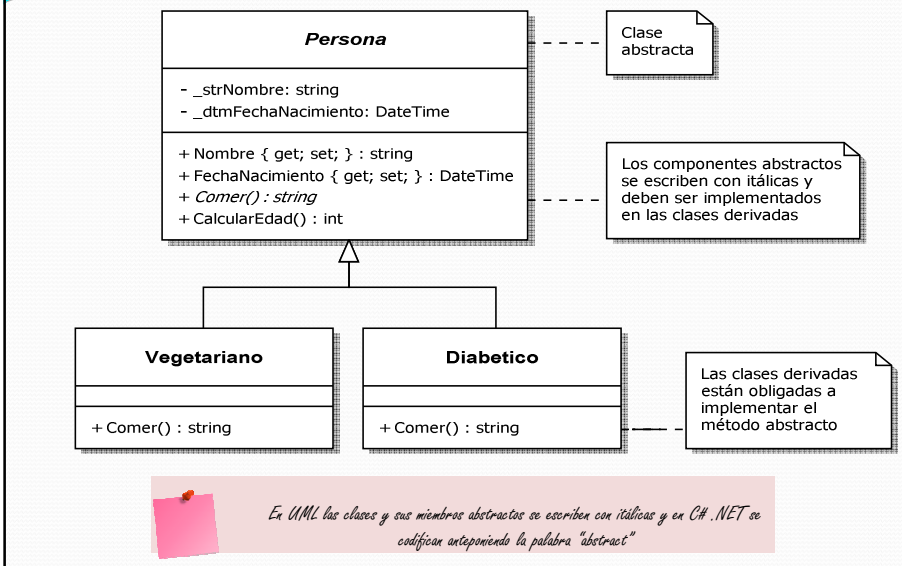
Implementación

Clase abstracta con elementos abstractos (cont.)

```
class Programa
{
    static void Main()
    {
        Empleado unEmpleado = new Empleado();
        unEmpleado.Nombre = "Juan";
        unEmpleado.Apellido = "Gonzalez";
        unEmpleado.Clave = 1;
        System.Console.WriteLine( unEmpleado.ConsultarTodosLosDatos() );
        System.Console.WriteLine( unEmpleado.ObtenerNombreCompleto() );

        Cliente unCliente = new Cliente();
        unCliente.Nombre = "Pedro";
        unCliente.Apellido = "Ramirez";
        unCliente.Clave = 34;
        System.Console.WriteLine( unCliente.ConsultarTodosLosDatos() );
        System.Console.WriteLine( unCliente.ObtenerNombreCompleto());
        System.Console.ReadLine();
    }
}
```

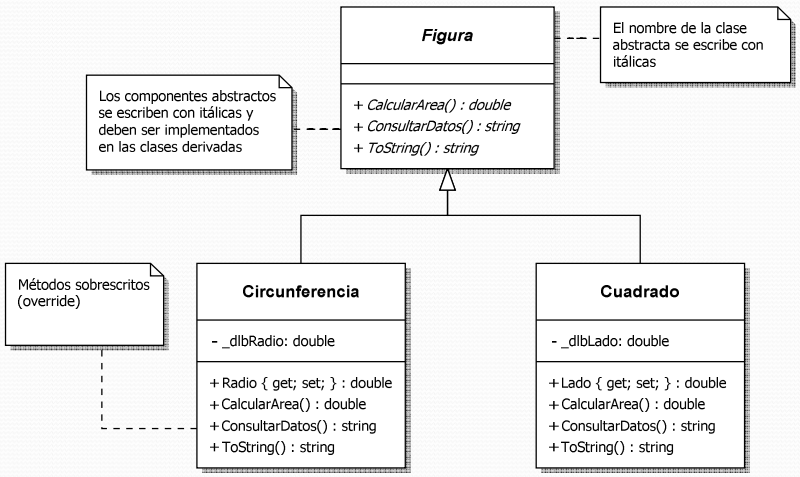
15

Miembros abstractos



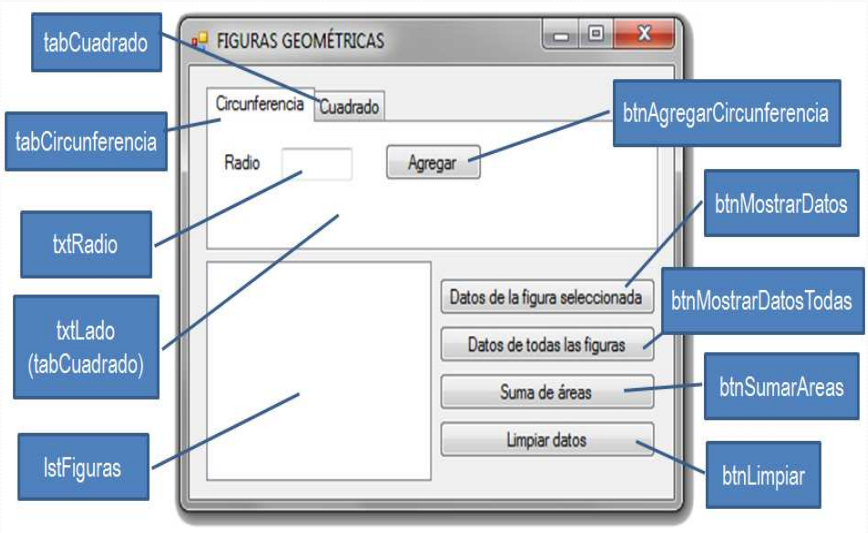
16

Prog. 5.5.- Clase abstracta con métodos abstractos



17

Diseño de la forma

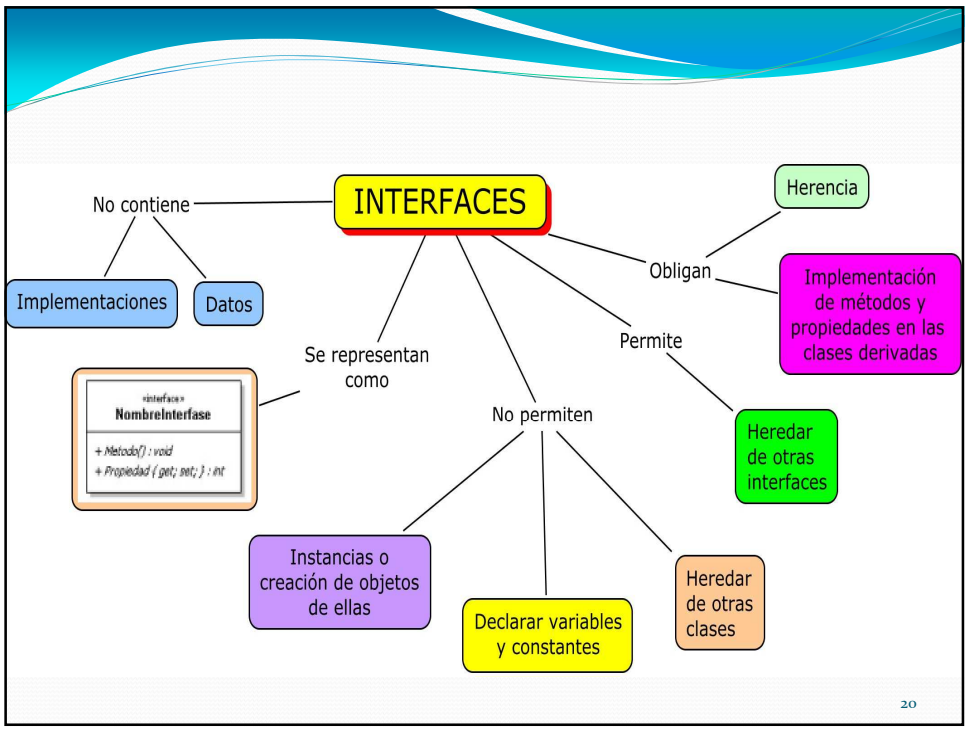


18

Interfaces

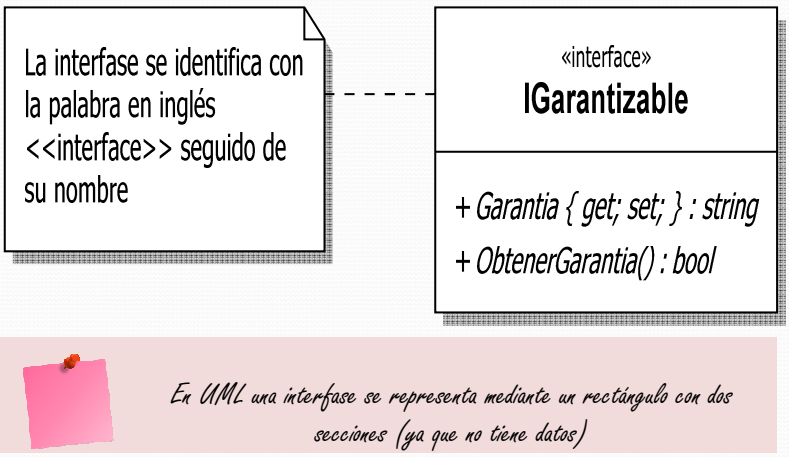
- Son mecanismos para que puedan interactuar varios objetos no relacionados entre sí
- Son protocolos o "contratos" que obligan la herencia
- Contienen las declaraciones de los métodos, pero no su implementación.
- Al igual que las clases abstractas, son plantillas de comportamiento que deben ser implementados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

19



20

Ejemplo de una interfase



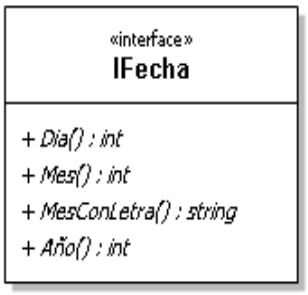
21

Notas acerca de las interfaces

- Una clase que herede de una interfase **debe** implementar **todas** las definiciones contenidas en ella.
- Los elementos de la interfase no llevan los modificadores "`public`" o "`abstract`".
- TODOS los elementos declarados dentro de una intefase se consideran públicos y abstractos.

22

Ejemplo de una interfase diseñada por el programador

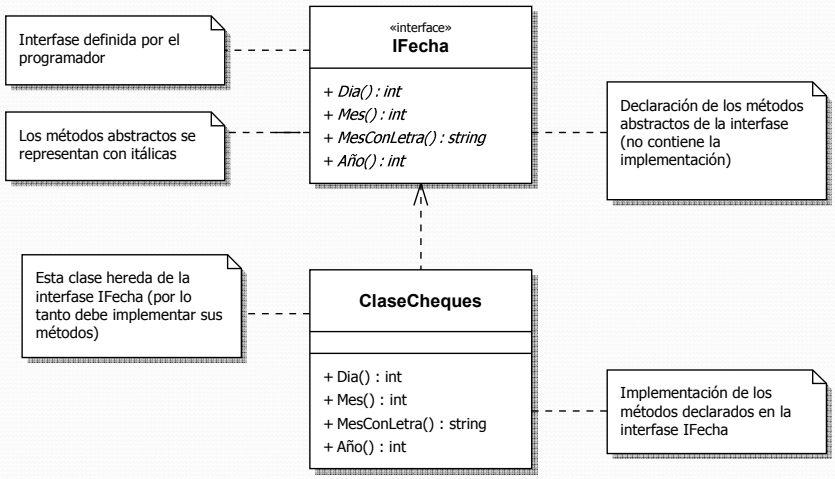


- Obtiene la fecha del sistema mediante `DateTime.Now`
- Obliga a implementar los métodos que contiene en las clases derivadas de ella.
- Se deben implementar **todos** sus métodos, de lo contrario nos indica un error.

Los miembros de una interfase se consideran públicos y virtuales por defecto (no hay necesidad de especificarlo)

23

Uso de una interfase diseñada por el programador



24

Declaración de la interfase

```
interface IFecha
{
    int Dia();
    int Mes();
    string MesConLetra();
    int Año();
}
```

25

Uso de una interfase en C#

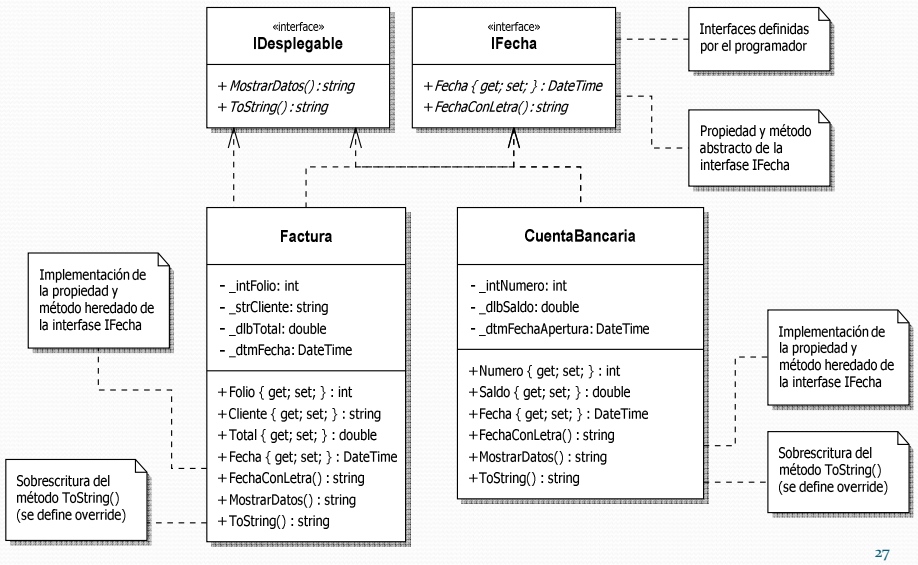
```
class ClaseCheques : IFecha
    {
        // Implementación de los métodos de la interfaz IFecha

        public int Dia( )
        {
            return DateTime.Now.Day;
        }

        public int Mes( )
        {
            return DateTime.Now.Month;
        }

        public int Año( )
        {
            return DateTime.Now.Year;
        }

        public string MesConLetra( )
        {
            switch ( Mes( ) )
            {
                case 1: return ("Enero"); break;
                case 2: return ("Febrero"); break;
                 …
                case 12: return ("Diciembre"); break;
            }
        }
    }
```

La ClaseCheques hereda de la interfase IFecha

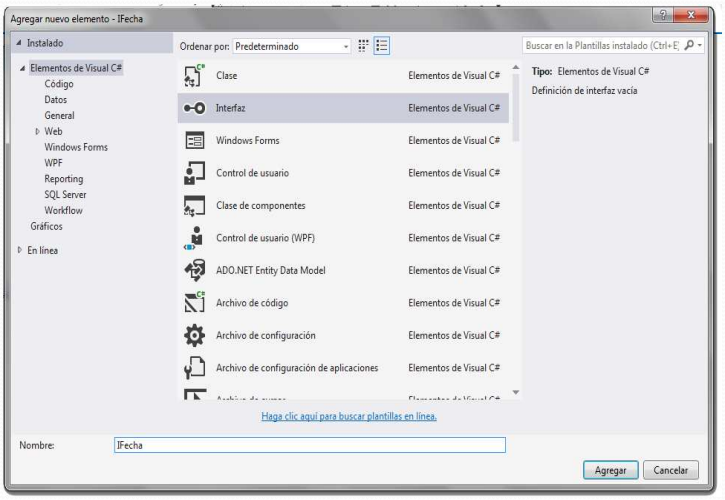
Implementación de los métodos de la interfase IFecha

26

Otro ejemplo de uso de interfaces



¿Cómo agregar una interfase al proyecto?



Ejemplo de una interfase

```
interface IVehiculo                    //Declaraciones solamente
{
    string Marca
    {
        get;
        set;
    }
    void Arrancar();
}
class Carro: IVehiculo                 //Implementación de toda la interfase
{
    private string _strMmarca;
    public string Marca
    {
        get { return _strMarca; }
        set { _strMarca = value;  }
    }
    public void Arrancar()
    {
        System.Console.WriteLine("Arrancar....Clase Carro");
    }
}
```

29

Ejemplo: Heredando de una clase e implementando dos interfaces

```
interface ICuadrado
{
    double Lado
    {
        get;
        set;
    }
}

interface IFiguraOperaciones
{
    double CalcularArea();
    double CalcularPerimetro();
}

public class Figura
{
    public virtual string ConsultarDatos()
    {
        return "Datos de la Figura: ";
    }
}
```

```
class Cuadrado :  Figura, ICuadrado,
                  IFiguraOperaciones
   {
       private double _dblLado;
       public double Lado
       {
           get { return _dblLado;  }
           set { _dblLado = value; }
       }
       public double CalcularArea()
       {
           return Lado * Lado;
       }
       public double CalcularPerimetro()
       {
           return 4 * Lado;
       }
       public override string ConsultarDatos()
       {
           return " Datos : Lado = " + Lado;
       }
   }
```

30

Ejemplo: Heredando de una clase e implementando dos interfaces (cont.)

```
class Program
   {
       static void Main()
       {
           Cuadrado c = new Cuadrado();
           c.Lado = 2;
           System.Console.WriteLine( c.ConsultarDatos() );
           System.Console.WriteLine("Area: " + c.CalcularArea());
           System.Console.WriteLine("Perimetro: " + c.CalcularPerimetro());
           System.Console.ReadLine();
       }
   }
```

31

Interfaces en C#

- IComparable
- IEquatable
- IEnumerator
- IEnumerable
- INotifyPropertyChanged
- Y otras …

32

La interfase 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

33

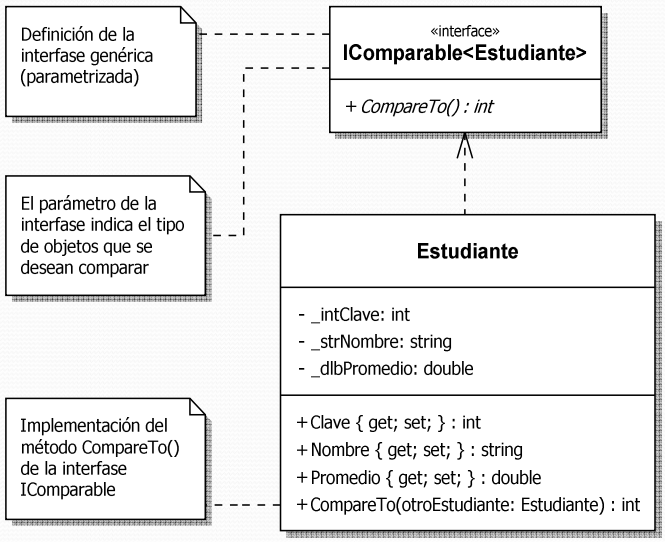
La función CompareTo()

`int CompareTo(obj)`

`-1`	Si this < obj
`0`	Si this == obj
`1`	Si this > obj

34

Uso de IComparable



35

Implementación de IComparable

```
class Estudiante : IComparable<Estudiante>
   {
       // Atributos privados
       private int _intClave;
       private string _strNombre;
       private double _dlbPromedio;

       // Propiedades públicas
       public int Clave
       {
           get { return _intClave; }
           set { _intClave = value; }
       }

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

       public double Promedio
       {
           get { return _dlbPromedio; }
           set { _dlbPromedio = value; }
       }

       // Implementación del método CompareTo de la interfase IComparable
       public int CompareTo(Estudiante otroEstudiante)
       {
           // Se utiliza el promedio de los estudiantes para determinar
           // el orden
           if (this.Promedio > otroEstudiante.Promedio)
               return (1);
           else
               if (this.Promedio < otroEstudiante.Promedio)
                   return (-1);
               else
                   return (0);
       }
   }
```

36

¿Cómo comparar datos de tipo *string*?

```
class Estudiante : IComparable<Estudiante>
   {
       // Atributos privados
       private int _intClave;
       private string _strNombre;
       private double _dlbPromedio;

       // Propiedades públicas
       public int Clave
       {
           get { return _intClave; }
           set { _intClave = value; }
       }

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

       public double Promedio
       {
           get { return _dlbPromedio; }
           set { _dlbPromedio = value; }
       }

       // Implementación del método CompareTo de la interfase IComparable
       public int CompareTo(Estudiante otroEstudiante)
       {
           return(this.Nombre.CompareTo(otroEstudiante.Nombre));
       }
   }
```

37

¿Un CompareTo() dentro de otro?

```
class Estudiante : IComparable<Estudiante>
   {

       . . .
       . . .
       . . .

       // Implementación del método CompareTo de la interfase Icomparable

       public int CompareTo(Estudiante otroEstudiante)
       {
           return(this.Nombre.CompareTo(otroEstudiante.Nombre));
       }
   }
```

- El `CompareTo()` de la clase `Estudiante` invoca al `CompareTo()` de la clase *`string`* (puesto que el `Nombre` es un dato de tipo cadena).

38

La interfase IEquatable

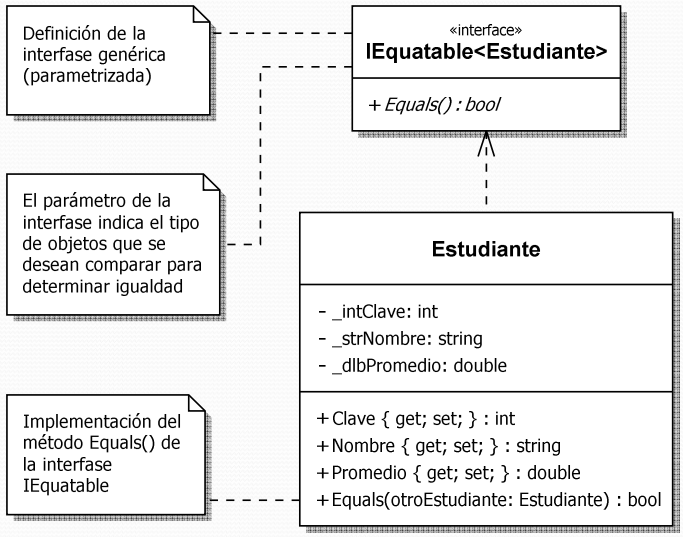
- Contiene la declaración del método `Equals()`

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

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

39

Uso de IEquatable



40

Implementación de IEquatable

```
class Estudiante : IEquatable<Estudiante>

   {
       // Atributos privados
       private int _intClave;
       private string _strNombre;
       private double _dlbPromedio;

       // Propiedades públicas
       public int Clave
       {
           get { return _intClave; }
           set { _intClave = value; }
       }

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

       public double Promedio
       {
           get { return _dlbPromedio; }
           set { _dlbPromedio = value; }
       }

       // Implementación del método Equals de la interfase IEquatable
       public bool Equals(Estudiante otroEstudiante)
       {
           // Se utiliza la clave de los estudiantes para determinar
           // si dos objetos son iguales
           return (this.Clave == otroEstudiante.Clave);
       }
   }
```

41

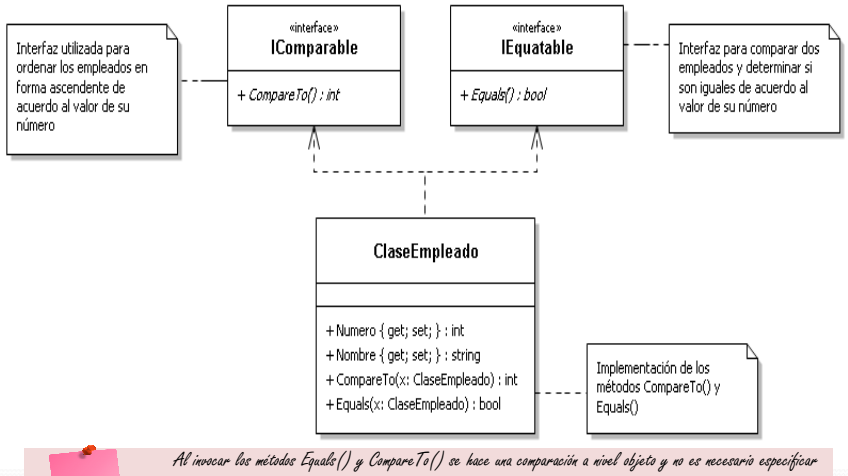
NOTA IMPORTANTE

- Las interfaces `IEquatable` e `IComparable` solamente comparan objetos del **mismo tipo**.
- No se pueden comparar objetos de diferentes tipos; es decir, creados a partir de clases diferentes.

Puesto que los objetos pueden realizar acciones, entonces tienen la capacidad de compararse entre sí para determinar si son iguales o para definir un orden específico a través de las interfaces IEquatable e IComparable

42

Uso de las interfaces IComparable e IEquatable



Al invocar los métodos Equals() y CompareTo() se hace una comparación a nivel objeto y no es necesario especificar cuál es el dato que se utiliza para hacer la comparación. Es la implementación de dichos métodos en la clase la que determina el criterio de comparación.

43

Implementación

```
class ClaseEmpleado: IComparable<ClaseEmpleado>, IEquatable<ClaseEmpleado>
{
    public int Numero { get; set; }
    public string Nombre { get; set; }

    public int CompareTo(ClaseEmpleado otroEmpleado)
    {
        return(this.Nombre.CompareTo(otroEmpleado.Nombre);
    }

    public bool Equals(ClaseEmpleado otroEmpleado)
    {
        return (this.Numero == otroEmpleado.Numero);
    }
}
```

La ClaseEmpleado Implementa las interfaces IComparable e IEquatable

Implementación de los métodos de las interfaces

Use el método CompareTo() para comparar datos de tipo string

44

Ejemplo de uso

- Declaración e inicialización de los objetos:

```
ClaseEmpleado miSecretaria = new ClaseEmpleado();
ClaseEmpleado miIntendente = new ClaseEmpleado();

miSecretaria.Numero = 2;
miSecretaria.Nombre = "Rosa";

miIntendente.Numero = 3;
miIntendente.Nombre = "Luis";
```

45

Ejemplo de uso (cont.)

```
int intResultado = miSecretaria.CompareTo(miIntendente);
switch (intResultado)
{
     case -1: MessageBox.Show("El nombre de la Secretaria es menor que el nombre del
Intendente");
                   break;
     case 0: MessageBox.Show("El nombre de la Secretaria es igual que el nombre del
Intendente");
                  break;
     case 1: MessageBox.Show("El nombre de la Secretaria es mayor que el nombre del
Intendente");
                  break;
}

if (miSecretaria.Equals(miIntendente))
    MessageBox.Show("El número de la Secretaria es igual que el número del
Intendente");
else
    MessageBox.Show("El número de la Secretaria es diferente que el número del
Intendente");
```

46

Restricciones de tipos de parámetros

- Se usan al diseñar clases genéricas
- Se utilizan para "obligar" a enviar ciertos tipos como parámetros a las clases genéricas
- Se implementan usando la sentencia `where`

47

Restricciones de tipos de parámetros (*cont.*)

Restricción	Descripción
where T: struct	El argumento de tipo debe ser un tipo de valor. Se puede especificar cualquier tipo de valor excepto Nullable. Para obtener más información, consulte Utilizar tipos que aceptan valores NULL.
where T : class	El argumento de tipo debe ser un tipo de referencia; esto se aplica también a cualquier tipo de clase, interfaz, delegado o matriz.
where T : new()	El argumento de tipo debe tener un constructor público sin parámetros. Cuando se utiliza la restricción `new()` con otras restricciones, debe especificarse en último lugar.
where T : <nombre de clase base>	El argumento de tipo debe ser la clase base especificada, o bien debe derivarse de la misma.
where T: <nombre de interfaz>	El argumento de tipo debe ser o implementar la interfaz especificada. Se pueden especificar varias restricciones de interfaz. La interfaz con restricciones también puede ser genérica.
where T : U	El argumento de tipo proporcionado para T debe ser o derivar del argumento proporcionado para U.

48

La restricción *where*

- Obliga a que una clase genérica utilice alguna restricción particular. P. ejem:
  - Una estructura
  - Una clase
  - Un constructor default
  - La implementación de una interfase
  - Una clase base o derivada
- Si no la contiene… ERROR !!!

49

Ejemplo de uso de *where*

- Un restaurante de comida rápida ofrece los siguientes productos:
  - Hamburguesas
  - Pizzas
  - Tacos
  - Tortas
  - Papas fritas
- Una motocicleta reparte cada alimento en su respectiva caja

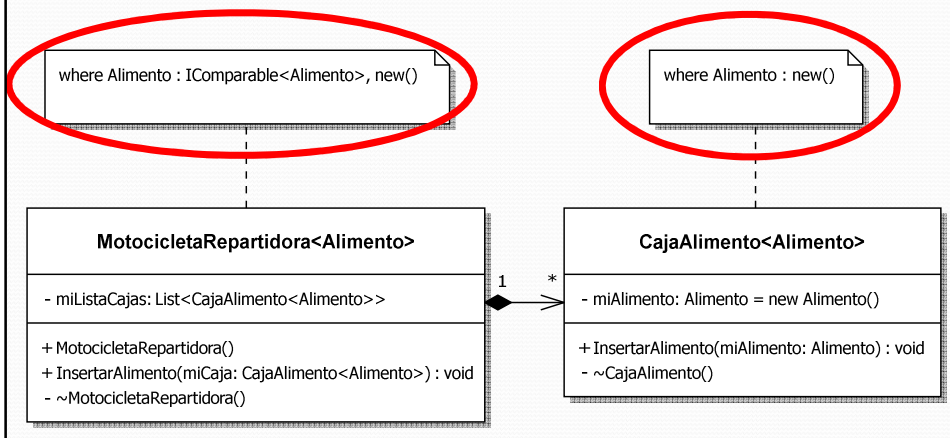
50

Ejemplo de uso de *where (cont.)*

- Cada alimento tiene los siguientes datos:
  - Nombre del alimento
  - Precio
  - Domicilio de entrega
  - Distancia en metros desde el restaurante
- La motocicleta transporta varias cajas con alimentos a la vez y las reparte de acuerdo a la distancia en metros (iniciando por el más cercano).

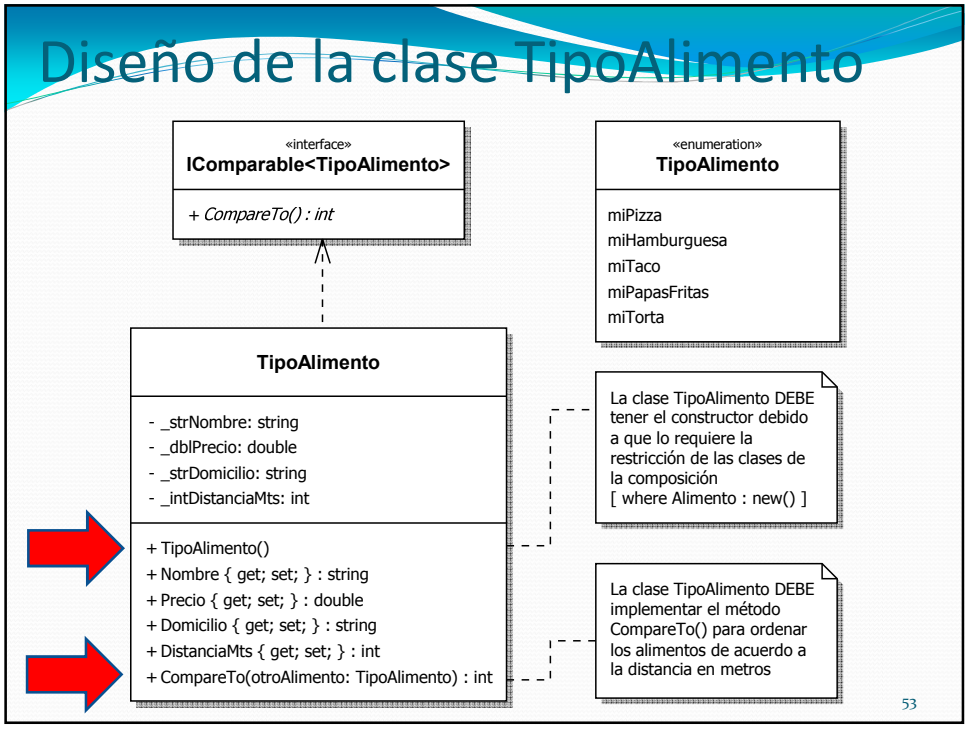
51

Diseño de la composición usando la sentencia *where*



52

Diseño de la clase TipoAlimento



La clase CajaAlimento

```
public class CajaAlimento<Alimento> where Alimento:new()
{
   private Alimento miAlimento;

   public CajaAlimento() {
      miAlimento = new Alimento();
   }

   public void InsertarAlimento(Alimento miAlimento) {
      this.miAlimento = miAlimento;
   }

   ~CajaAlimento() {
      // Elimina el objeto miAlimento
      miAlimento = default(Alimento);
   }
}
```

54

La clase MotocicletaRepartidora

```
class MotocicletaRepartidora<Alimento> where Alimento :
IComparable<Alimento>, new()
{
   private List<CajaAlimento<Alimento>> miListaCajas;

   public MotocicletaRepartidora() {
      miListaCajas = new List<CajaAlimento<Alimento>>();
   }

   public void InsertarAlimento(CajaAlimento<Alimento> miCaja) {
      miListaCajas.Add(miCaja);
   }

   ~MotocicletaRepartidora() {
      miListaCajas.Clear();
   }
}
```

55

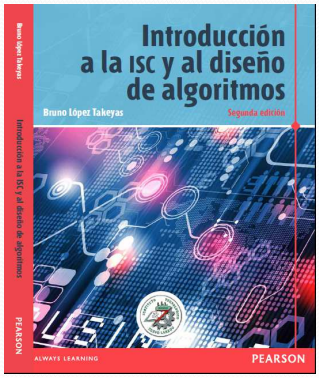
Implementación de la sentencia *where*

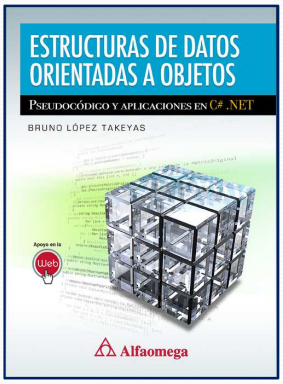
- Cada alimento **DEBE** implementar el método `CompareTo()` de la interfase `IComparable`
- El método `CompareTo()` compara los alimentos y los ordena de acuerdo a la distancia de la entrega
- La sentencia `where` de la clase `MotocicletaRepartidora` **OBLIGA** a que todos los objetos de su composición implementen el método `CompareTo()`

56

Otros títulos del autor

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





takeyas@itnuevolaredo.edu.mx

Bruno López Takeyas