UNIVERSIDAD AUTÓNOMA DE SINALOA

Facultad de Informática Culiacán

POO en C#

Instructor:

MC. Gerardo Gálvez Gámez

gerardo.galvez@uas.edu.mx

Octubre de 2017

Materia del 2do Semestre

ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

2.- Unidad I Conceptos Básicos.pdf

5 .-Unidad IV El modelo del Diseño final.pdf

Programación Orientada a Objetos

La Sintaxis en el Lenguaje C#

POO en C#• FIUAS

Competencia del Tema

Al final del tema, los estudiantes serán capaces de:

Conocer y aplicar la sintaxis de la programación Orientada a Objetos, aplicada en el lenguaje **C#,** en la solución de diversos problemas**.**

El mundo color de Objetos

- ¿Por qué Orientación a Objetos (OO)?
  - Se parece más al mundo real
  - Permite representar modelos complejos
  - Muy apropiada para aplicaciones de negocios
  - Las empresas ahora sí aceptan la OO
  - Las nuevas plataformas de desarrollo la han adoptado (Java / .NET)
- Permite representar de manera relativamente simple modelos y realidades muy complejas y esto hace que el software sea más fácil de *programar, comprender y mantener.*

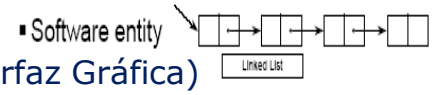
Pensar en Objetos ….

¿Qué es un Objeto?

- Informalmente, un objeto representa una entidad del mundo real
- Entidades Físicas
  - (Ej.: Vehículo, Casa, Producto)

- Entidades Conceptuales
  - (Ej.: Proceso Químico, Transacción Bancaria)

- Entidades de Software
  - (Ej.: Lista Enlazada, Interfaz Gráfica)



¿Qué es un Objeto?

- Definición Formal (James Rumbaugh y Grady Booch):
  - "un objeto es una abstracción de la realidad que tiene un significado concreto y claro para el problema que se está modelando". Ejemplo: "un Auto"

- Un objeto posee (Booch):
  - Identidad:
    - Los objetos se distinguen unos de otros



  - Comportamiento:
    - Los objetos pueden realizar tareas
  - Estado:
    - Los objetos contienen información

POO en C#• FIUAS

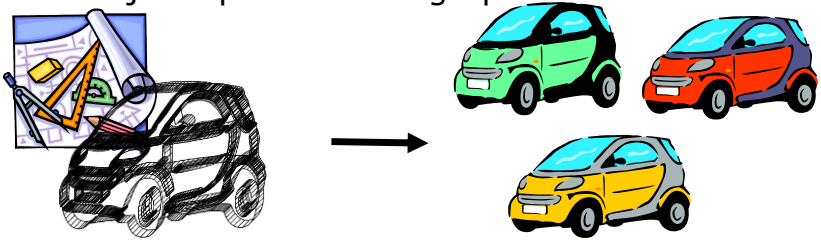
¿Qué es una Clase?

- Una clase es una descripción de un grupo de objetos con:
  - Propiedades en común (Identidades ó atributos)
  - Comportamiento similar (Operaciones)
  - La misma forma de relacionarse con otros objetos (relaciones)
  - Una semántica en común (significan lo mismo)
- Una clase es una abstracción que:
  - Enfatiza las características relevantes
  - Suprime otras características (simplificación)
- Un objeto es una instancia de una clase

Desempleado
- Nombre : string - Dirección : string - Teléfono : string - Email: string - Curriculum : string
+ Alta():void + Modificar():void + ConsultarDatos():void + Eliminar():void + Postularse Vacante():void

Objetos y Clases

- Una clase es una definición abstracta de un objeto
  - Define la estructura y el comportamiento compartidos por los objetos
  - Sirve como modelo para la creación de objetos
- Los objetos pueden ser agrupados en clases



POO en C#• FIUAS

Creación y Destrucción de Objetos

- Uso de Constructores
- Objetos y Memoria

Uso de constructores

- Creación de objetos
- Uso del constructor por defecto
- Sustitución del constructor por defecto
- Sobrecarga de constructores

POO en C#• FIUAS

Creación de objetos

- Paso 1: Asignación de memoria
  - Se usa **new** para asignar memoria desde el montón
- Paso 2: Inicialización del objeto usando un constructor
  - Se usa el nombre de la clase seguido por paréntesis

```
Circulo Figura = new Circulo( );
```

Uso del constructor por defecto

- Características de un constructor por defecto:
  - Acceso público
  - Mismo nombre que la clase
  - No tiene tipo de retorno (ni siquiera **void**)
  - No recibe ningún argumento
  - Inicializa todos los campos a **cero**, **false** o **null**
- Sintaxis del constructor:

```
class NombreClase
 {
        public NombreClase( )
        {
                ...
        }
 }
```

POO en C#• FIUAS

Sustitución del constructor por defecto

- El constructor por defecto puede no ser adecuado
  - En ese caso no hay que usarlo, sino escribir otro

```
class Circulo
{
        private float radio
        public Circulo( )
        {
                this.radio=0.0F;
        }
}
```

Sobrecarga de constructores

- Los constructores son métodos y pueden estar sobrecargados
  - Mismo ámbito, mismo nombre, distintos parámetros
  - Permite inicializar objetos de distintas maneras
- AVISO
  - Si se escribe un constructor para una clase, el compilador no creará un constructor por defecto

```
class Circulo
{
        private float radio
        public Circulo ( )
        {
            this.radio=0.0F;
        }
        public Circulo (float Radio )
        {
            this.radio=Radio;
        }
}
```

POO en C#• FIUAS

Los destructores

- Se utilizan para destruir instancias de clases.
- Suele ser útil para liberar recursos tales como los Archivos o las conexiones de redes abiertas que el objeto a destruir estuviese utilizando en el momento en que se fuese a destruir.
- Los destructores no se pueden definir en estructuras. Sólo se utilizan con clases.
- Una clase sólo puede tener un destructor.
- Los destructores no se pueden heredar ni sobrecargar.
- No se puede llamar a los destructores. Se invocan automáticamente.
- Un destructor no permite modificadores de acceso ni tiene parámetros.

La sintaxis que se usa para definir un destructor es la siguiente:

```
~ NombreClase()
{
        <código>
}
```

Objetos y memoria

- Tiempo de vida de un objeto
- Objetos y ámbito
- Recolección de basura

Tiempo de vida de un objeto

- Creación de objetos
  - Se usa **new** para asignar memoria
  - Se usa un constructor para inicializar un objeto en esa memoria
- Uso de objetos
  - Llamadas a métodos
- Destrucción de objetos
  - Se vuelve a convertir el objeto en memoria
  - Se libera la memoria

Objetos y ámbito

- El tiempo de vida de un valor a local está vinculado al ámbito en el que está declarado
  - Tiempo de vida corto (en general)
  - Creación y destrucción deterministas
- El tiempo de vida de un objeto dinámico no está vinculado a su ámbito
  - Tiempo de vida más largo
  - Destrucción no determinista

Recolección de basura

- No es posible destruir objetos de forma explícita
  - C# no incluye un inverso de **new** (como **delete**)
  - Ello se debe a que una función de eliminación explícita es una importante fuente de errores en otros lenguajes
- Los objetos se destruyen por recolección de basura
  - Busca objetos inalcanzables y los destruye
  - Los convierte de nuevo en memoria binaria no utilizada
  - Normalmente lo hace cuando empieza a faltar memoria
  - puede forzarse llamando al método Collect() de la clase System.GC

Liberación explícita de recursos

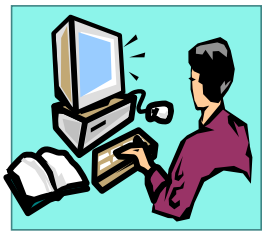
- Para liberar explícitamente el recurso antes de que el recolector de elementos no utilizados libere el objeto.
- Para ello debe implementar un **método Dispose** desde la interfaz IDisposable que realiza la limpieza del objeto necesaria.

- Más detalles:

Limpiar recursos no administrados
Implementar un método Dispose
using (Instrucción, Referencia de C#)

Práctica – Creación de objetos

Crear objetos en diferentes ámbitos y probar su accesibilidad.



Actividad #1

- Elaborar un proyecto, orientado a objetos, que permita realizar las siguientes operaciones sobre un Circulo:
  - Calcular su Área: Se utiliza la siguiente formula:
    - Area: PI * radio^2
  - Calcular su Longitud: Se emplea la siguiente formula:
    - Longitud: 2 * PI *radio

Diseño de clase propuesto

Circulo
-radio:float
+Circulo() +Circulo(pRadio:float) + Radio:float (get, set) +Area():double +Longitud():double

Actividad #2: Diseñar y programar la Clase, e instanciar objetos.

Elaborar un proyecto, orientado a objetos, que permita calcular el sueldo quincenal y mensual de un empleado.

El empleado cuenta con los siguientes datos: ***RFC, Nombre, Puesto:***

El sueldo Mensual, se calcula en base a la siguiente formula:

- Mensual=Sueldo Base – Retención – Cuota IMSS

Donde: La retención de calcula en base a:

- Para un SueldoBase de hasta de 1500 no hay retención,
- Para un SueldoBase mayor de 1500 y hasta 3000 el porcentaje de retención es de 5%.
- Para un SueldoBase mayor a 3000 el porcentaje de retención es el 8%.
- Nota: el sueldo quincenal es la mitad del sueldo mensual.

Análisis y Diseño Orientado a Objetos

Materia Relacionada: AYDOO

Diseño de Clases (jerarquía de clases)

Empleado	
-rfc:string -nombre:string -puesto:string -sueldobase:float -cuotaIMSS:float	Atributos(campos)
+Empleado() +Empleado(pRFC:string, pNombre:string, pPuesto:string, pSueldoBase:float, pCuotaIMSS:float)	Constructores
+RFC:string (get,set) +Nombre:string(get,set) +Puesto:string(get,set) +SueldoBase:float(get,set) +CuotaIMSS:float(get,set)	Propiedades
-Retencion():float +SueldoMensual():float +SueldoQuincenal():float	Métodos

Análisis del método Retención

Materia Relacionada: Algoritmia

POO en C#• FIUAS

Análisis del problema (Método Retención)

1. **Información de Salida**
   - Retencion

2. **Datos Conocidos**
   - PorcentajeRetencion1=5
   - PorcentajeRetencion2=8
   - ValorRetencion1=1500
   - ValorRetencion2=3000

3. **Datos no Conocidos (Atributos del Objeto)**
   - SueldoBase

4. **Restricciones**
   - No se debe solicitar al usuario el valor de la retención.
   - El sueldo base y la retención están expresados en moneda mexicana.

Proceso

Escoger y decidir las operaciones a efectuar.

- **Paso#1: Hacer que Retencion tome el valor 0, Sí el valor de SueldoBase es menor al ValorRetencion1.**
  - **Retencion=0**
- **Paso #2: De no cumplirse la condición del paso #1, hacer que Retencion tome el valor correspondiente al PorcentajeRetencion1, Si SueldoBase es Menor a ValorRetencion2.**
  - **Retencion= SueldoBase*(PorcentajeRetencion1/100)**
- **Paso #3: De no cumplirse la condición del paso #2, hacer que Retencion tome el valor correspondiente al PorcentajeRetencion2:**
  - **Retencion= SueldoBase*(PorcentajeRetencion2/100)**

Construcción del *Algoritmo (Pseudocódigo)*

Objetivo: Determinar retención correspondiente a un empleado.
Programador: MC. Gálvez Gámez Gerardo
Fecha: __/Octubre/2014

//atributo de clase
SueldoBase

```
REAL  Retencion()
INICIO
    //Definición de Variables y Constantes
     CONST REALPorcentajeRetencion1=5, PorcentajeRetencion2=8
     CONST REAL ValorRetencion1=1500, ValorRetencion2=3000
     REAL Retencion
     //Proceso determinar el tipo de número
   SI SueldoBase <ValorRetencion1 ENTONCES
          Retencion=0
   SI_NO
       SI SueldoBase <ValorRetencion2 ENTONCES
          Retencion=SueldoBase * (PorcentajeRetencion1/100)
     SI_NO
          Retencion=SueldoBase * (PorcentajeRetencion2/100)
     FIN_SI
    FIN_SI
     //(salida)
     REGRESAR Retencion
FIN
```

Actividad #3: Diseñar y programar la Clase, e instanciar objetos.

Elaborar un proyecto, orientado a objetos, que permita realizar operaciones entre conjuntos (al menos dos).

Matemáticas Discretas — Lógica Matemática

SUBCONJUNTO

Un conjunto A es **subconjunto** de un conjunto B, si cada elemento del conjunto A es también elemento de un conjunto B

Materia Relacionada: Matemáticas Discretas

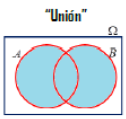
Actividad #4: Diseñar y programar la Clase, e instanciar objetos.

Elaborar un proyecto, orientado a objetos, que permita realizar operaciones entre conjuntos (al menos dos).

UNIÓN

La unión de A y B es el conjunto formado por todos los elementos que pertenecen a A o a B Se denota como: $A \cup B$

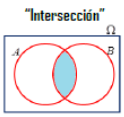
$$A \cup B = \{x | x \in A \vee x \in B\}$$



INTERSECCIÓN

La intersección de A y B es el conjunto formado por todos los elementos que pertenecen a A y a B Se denota como: $A \cap B$

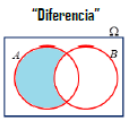
$$A \cap B = \{x | x \in A \wedge x \in B\}$$



DIFERENCIA

La diferencia entre dos conjuntos A y B es el conjunto formado por todos los elementos que pertenecen a A y no pertenecen a B. Se denota por $A - B$ se lee "A menos B"

$$A - B = \{x | x \in A \wedge x \notin B\}$$



Materia Relacionada: Matemáticas Discretas

Actividad #5: Diseñar y programar la Clase, e instanciar objetos.

Elaborar un proyecto, orientado a objetos, que permita representar dos matrices de dimensiones N x M y calcule la sumatoria y resta .

$$A=\begin{pmatrix}2 & -6 & 10\\ 6 & 4 & 7\\ 1 & -3 & 6\end{pmatrix} \qquad B=\begin{pmatrix}-5 & 4 & 0\\ -5 & 3 & 2\\ 4 & 5 & -7\end{pmatrix}$$

a. $A-B=$

b. $C+D=$

Materia Relacionada: Matemáticas II

Actividad #6: Diseñar y programar la Clase, e instanciar objetos.

Matrices

I. En base a las matrices definidas realiza las operaciones indicadas.

$$A=\begin{pmatrix}2 & -6 & 10\\ 6 & 4 & 7\\ 1 & -3 & 6\end{pmatrix} \quad B=\begin{pmatrix}-5 & 4 & 0\\ -5 & 3 & 2\\ 4 & 5 & -7\end{pmatrix} \quad C=\begin{pmatrix}-5 & -2 & -1 & 4\\ 4 & 4 & 0 & -7\\ 9 & 7 & 1 & 8\\ 11 & 3 & 4 & 3\end{pmatrix} \quad D=\begin{pmatrix}4 & 7 & 2 & 1\\ 8 & -5 & 9 & 5\\ 3 & 4 & 3 & 6\\ 0 & 2 & -5 & 4\end{pmatrix}$$

a. $A-B=$

b. $C+D=$

c. $AB=$

d. $BC=$

e. $DC=$

f. $D^t=$

g. $C^t=$

h. $(C+D)^t=$

Materia Relacionada: Matemáticas II

Herencia en C#

En .NET solo se permite Herencia Simple

POO en C#• FIUAS

Notas generales

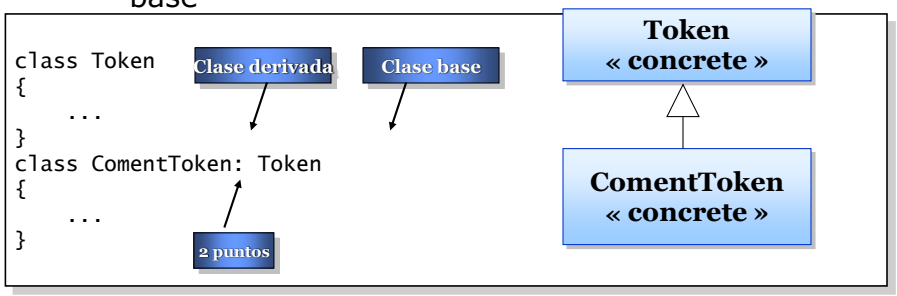
- Derivación de clases
- Implementación de métodos
- Uso de clases selladas
- Uso de interfaces
- Uso de clases abstractas

Derivación de clases

- Extensión de clases base
- Acceso a miembros de la clase base
- Llamadas a constructores de la clase base

Extensión de clases base

- Sintaxis para derivar una clase desde una clase base



- Una clase derivada hereda la mayor parte de los elementos de su clase base
- Una clase derivada no puede ser más accesible que su clase base

Acceso a miembros de la clase base

```
class Token
{   ...
    protected string name;
}
class ComentToken: Token
{   ...
    public string Name( )
    {
        return name;
    }
}
```

✓

```
class Outside
{
    void Fails(Token t)
    {
       ...
      t.name
       ...
  }
}
```

✘

- Los miembros heredados con protección están implícitamente protegidos en la clase derivada
- Los miembros de una clase derivada sólo pueden acceder a sus miembros heredados con protección

Llamadas a constructores de la clase base

- Las declaraciones de constructores deben usar la palabra base
- Una clase derivada no puede acceder a un constructor privado de la clase base
- Se usa la palabra base para habilitar el ámbito del identificador

```
class Token
{
   protected Token(string name) { ... }
   ...
}
class ComentToken: Token
{
   public ComentToken(string name) : base(name) { }
   ...
}
```

Implementación de métodos

- Definición de métodos virtuales
- Uso de métodos virtuales
- Sustitución de métodos (override)
- Uso de métodos override
- Uso de new para ocultar métodos
- Uso de la palabra reservada new

Definición de métodos virtuales

- Sintaxis: Se declara como virtual

```
class Token
{
    ...
    public int LineNumber( )
    { ...
    }
    public virtual string Name( )
    { ...
    }
}
```

Uso de métodos virtuales

- Para usar métodos virtuales:
  - No se puede declarar métodos virtuales como estáticos
  - No se puede declarar métodos virtuales como privados

POO en C#• FIUAS

Sustitución de métodos (override)

- Sintaxis: Se usa la palabra reservada override

```
class Token
{   ...
    public virtual string Name( ) { ... }
}
class ComentToken: Token
{   ...
    public override string Name( ) { ... }
}
```

Uso de métodos override

- **Sólo se sustituyen métodos virtuales heredados idénticos**

```
class Token
{   ...
    public int LineNumber( )  { ... }
    public virtual string Name( ) { ... }
}
class ComentToken: Token
{   ...
    public override int LineNumber( ) { ... }
    public override string Name( ) { ... }
}
```

- **Un método override debe coincidir con su método virtual asociado**
- **Se puede sustituir un método override**
- **No se puede declarar explícitamente un override como virtual**
- **No se puede declarar un método override como static o private**

Uso de new para ocultar métodos

- Esta técnica consiste en crear dentro de una clase derivada, miembros con el mismo nombre (y firma, en el caso de métodos) que los existentes en la clase base, pero ocultando el acceso a los miembros de la clase base para los objetos instanciados de la subclase.

```
class Token
{   ...
    public int LineNumber( )  { ... }
}
class ComentToken: Token
{   ...
    new public int LineNumber( )  { ... }

}
```

Uso de la palabra reservada new

- Ocultar tanto métodos virtuales como no virtuales

```
class Token
{   ...
    public int LineNumber( )  { ... }
    public virtual string Name( ) { ... }
}
class ComentToken: Token
{   ...
    new public int LineNumber( )  { ... }
    public override string Name( ) { ... }
}
```

- Resolver conflictos de nombre en el código
- Ocultar métodos que tengan signaturas idénticas

Uso de clases selladas

- Toda clase que declaremos en nuestro código es heredable por defecto; esto supone un elevado grado de responsabilidad, en el caso de que diseñemos una clase pensando en que pueda ser utilizada por otros programadores que hereden de ella.
- Si en un determinado momento, necesitamos hacer cambios en nuestra clase, dichos cambios afectarán a las clases derivadas que hayan sido creadas.
- Por dicho motivo, si no queremos que nuestra clase pueda ser heredada por otras, debemos declararla de forma que no permita herencia; a este tipo de clase se le denomina clase no heredable o sellada (sealed).

Uso de clases selladas

- Ninguna clase puede derivar de una clase sellada
- Las clases selladas sirven para optimizar operaciones en tiempo de ejecución
- Muchas clases de .NET Framework son selladas: String, StringBuilder, etc.
- Sintaxis: Se usa la palabra reservada sealed

```
namespace System
{
    public sealed class String
    {
        ...
    }
}
namespace Mine
{
    class FancyString: String { ... } ✘
}
```

POO en C#• FIUAS

Uso de interfaces

- Declaración de interfaces
- Implementación de varias interfaces
- Implementación de métodos de interfaz

Interfaces

- Recurso de diseño soportado por los lenguajes orientados a objetos que permite definir comportamientos
- Permite que clases que no están estrechamente relacionadas entre sí deban tener el mismo comportamiento
- La implementación de una interfaz es un contrato que obliga a la clase a implementar todos los métodos definidos en la interfaz

POO en C#• FIUAS

Interfaces

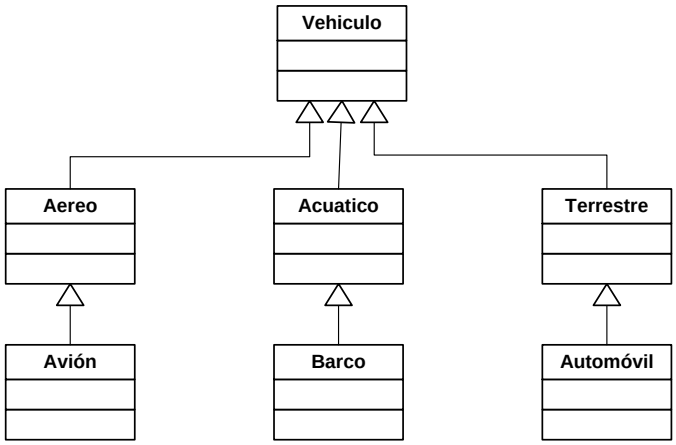
- Un interfaz proporciona, a modo de declaración, una lista de propiedades y métodos, que posteriormente serán codificados en una o varias clases.
- Debido a su naturaleza declarativa, un interfaz no contiene el código de los miembros que expresa; dicho código será escrito en las clases que implementen el interfaz.

Interfaces

- El concepto de interfaz es análogo al de contrato, las partes integrantes son el propio interfaz y la clase que lo implementa. Mientras que el interfaz no puede ser cambiado desde el momento en que sea implementado, la clase que lo implementa se compromete a crear la lista de miembros en la misma forma que indica el interfaz

- Los interfaces nos permiten definir conjuntos reducidos de funcionalidades, constituyendo una útil herramienta de cara al polimorfismo. El mismo interfaz, implementado en distintas clases, podrá tener a su vez código distinto, con lo que los objetos de diferentes clases que implementen un interfaz común, pueden tener un comportamiento diferente.

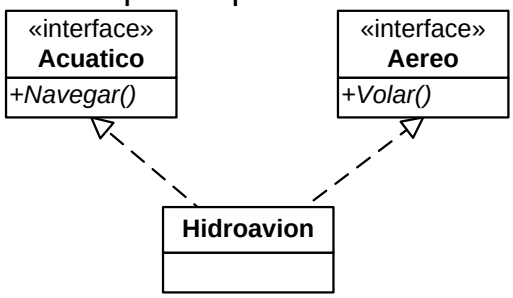
Interfaces



¿ De que clase heredaría la clase Hidroavión ?

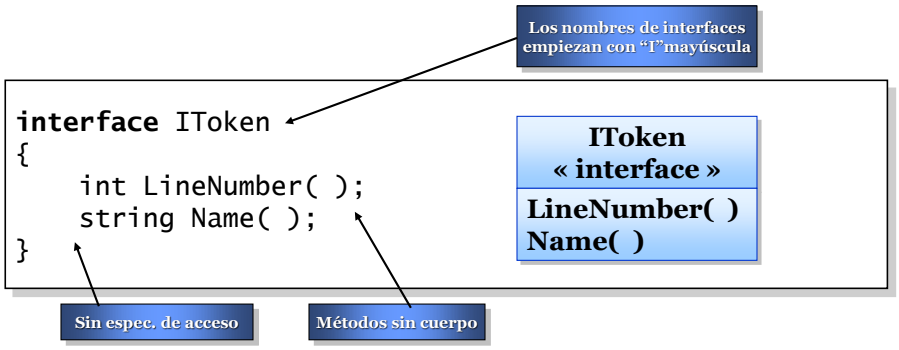
Interfaces

- Se crean las interfaces que definen comportamiento
- Hidroavión deberá definir los comportamientos de cada una de las interfaces que implemente



Declaración de interfaces

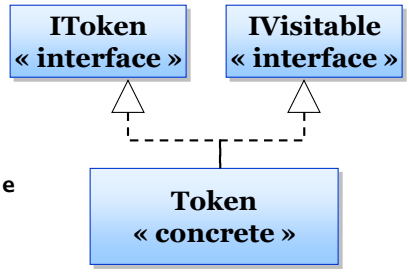
- Sintaxis: Para declarar métodos se usa la palabra reservada interface



Implementación de varias interfaces

- Una clase puede implementar cero o más interfaces

```
interface IToken
{
    string Name( );
}
interface IVisitable
{
    void Accept(IVisitante v);
}
class Token: IToken, IVisitable
{ ...
}
```



- Una interfaz puede extender cero o más interfaces

- Una clase puede ser más accesible que sus interfaces base
- Una interfaz no puede ser más accesible que su interfaz base
- Una clase implementa todos los métodos de interfaz heredados

Implementación de métodos de interfaz

- El método que implementa debe ser igual que el método de interfaz
- El método que implementa puede ser virtual o no virtual

```
class Token: IToken, IVisitable
{
    public virtual string Name( )
    { ...
    }
    public void Accept(IVisitante v)
    { ...
    }
}
```

Mismo acceso
Mismo retorno
Mismo nombre
Mismos parámetros

Uso de clases abstractas

- Las clases abstractas se emplean para proporcionar implementaciones parciales de clases que se completan con clases derivadas concretas. Las clases abstractas son especialmente útiles para la implementación parcial de una interfaz que puede ser reutilizada por varias clases derivadas.
  - Declaración de clases abstractas
  - Uso de clases abstractas en una jerarquía de clases
  - Comparación de clases abstractas e interfaces
  - Implementación de métodos abstractos
  - Uso de métodos abstractos

Declaración de clases abstractas

- Se usa la palabra reservada abstract

```
abstract class Token
{
    ...
}
class Test
{
    static void Main( )
    {
        new Token( );
    }
}
```

Token
{ abstract }

No se pueden crear instancias de una clase abstracta

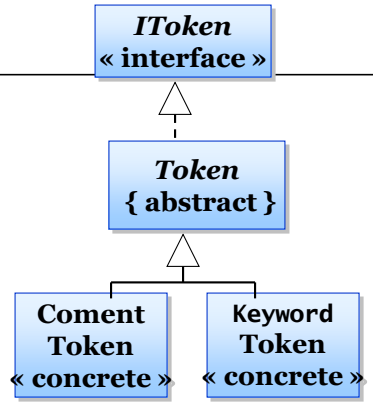
Uso de clases abstractas en una jerarquía de clases

Ejemplo 1 Las clases abstractas se emplean a menudo para la implementación parcial de interfaces.

```
interface IToken
{
    string Name( );
}

abstract class Token: IToken
{
    string IToken.Name( )
    {   ...
    }
    ...
}

class ComentToken: Token
{   ...
}
class KeywordToken: Token
{   ...
}
```



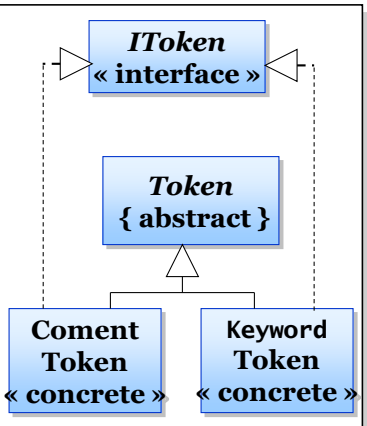
Uso de clases abstractas en una jerarquía de clases *(cont.)*

- **Ejemplo 2**

```
interface IToken
{
    string Name( );
}

abstract class Token
{
    public virtual string Name( )
    {   ...
    }
    ...
}

class ComentToken: Token, IToken
{   ...
}
class KeywordToken: Token, IToken
{   ...
}
```



Comparación de clases abstractas e interfaces

- Tanto las clases abstractas como las interfaces existen para derivar otras clases de ellas (o ser implementadas). Sin embargo, una clase puede extender como máximo una clase abstracta, por lo que hay que tener más cuidado cuando se deriva de una clase abstracta que cuando se deriva de una interfaz. Las clases abstractas se deben utilizar solamente para implementar relaciones del tipo "es un".

POO en C#• FIUAS

Comparación de clases abstractas e interfaces

- Parecidos
  - No se pueden crear instancias de ninguna de ellas
  - No se puede sellar ninguna de ellas
- Diferencias
  - Las interfaces no pueden contener implementaciones
  - Las interfaces no pueden declarar miembros no públicos
  - Las interfaces no pueden extender nada que no sea una interfaz

Implementación de métodos abstractos

- Sintaxis: Se usa la palabra reservada abstract

```
abstract class Token
{
    public virtual string Name( ) { ... }
    public abstract int Longitud( );
}
class ComentToken: Token
{
    public override string Name( ) { ... }
    public override int Longitud( ) { ... }
}
```

POO en C#• FIUAS

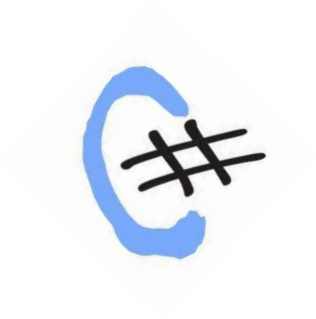
Uso de métodos abstractos

Antes de implementar un método abstracto hay que tener en cuenta sus características y las restricciones que impone.

- Los métodos abstractos son virtuales
- Los métodos override pueden sustituir a métodos abstractos en otras clases derivadas
- Los métodos abstractos pueden sustituir a métodos de la clase base declarados como virtuales
- Los métodos abstractos pueden sustituir a métodos de la clase base declarados como override
- Es obligatorio sustituir un método abstracto, pero no es necesario sustituir un método virtual.

Visual C#	Uso
ClaseDerivada : ClaseBase	Una clase derivada hereda los miembros de una clase base.
virtual	Declara que un miembro de la clase base puede ser sobrescrito en una clase derivada.
override	Declara que un miembro de la clase derivada sobreescribe el miembro del mismo nombre de la clase base.
new	Declara que un miembro de la clase derivada oculta el miembro del mismo nombre de la clase base.
abstract	Declara que una clase provee un templete para la clase derivada. Este tipo e clase es llamada una clase Abstracta, y no puede ser instanciada.
abstract	Declara que un miembro de la clase provee un templete para miembros derivados. Este tipo de miembro es llamado un miembro abstracto y no puede ser invocado.
base	Llama a los miembros de la clase base dentro de una clase derivada.
this	Llama a un miembro de la actual instancia de una clase.
interface	Crea una interface que define los miembros que una clase debe tener.
classname : interfacename	Usa una definición de interface en una clase.

POO en C#• FIUAS

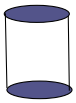


Figuras Geométricas (Luis joyanes Aguilar)

Circulo

Area: PI * radio²

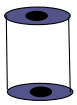
Longitud: 2 * PI *radio



Cilindro

Volumen: PI * radio² *altura

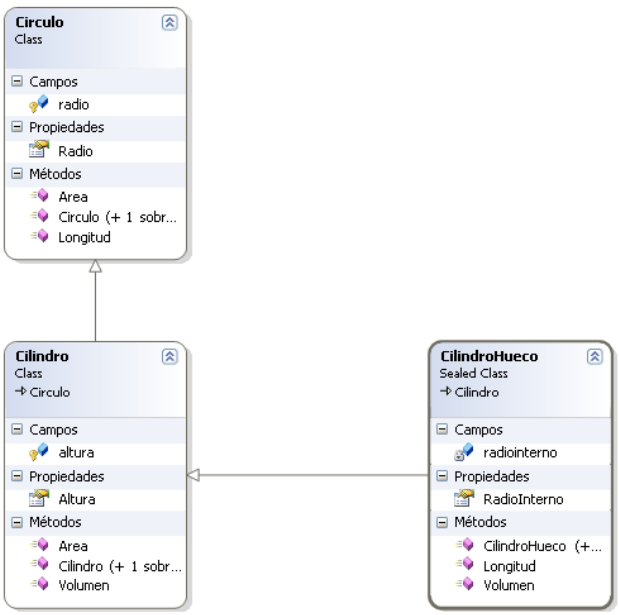
Area: 2 * PI * radio *altura + 2 * PI *radio²



Cilindro Hueco

Longitud: 2 * PI *(radio² - radioInterno²) + 2 * PI * radio * altura + 2*PI*altura*radioInterno

Volumen: PI *(radio² - radioInterno²) * altura



Preguntas?