PROGRAMACIÓN EN C# .NET

Programación Orientada a Objetos en C#

Ing. Bruno López Takeyas

Instituto Tecnológico de Nuevo Laredo

¿Qué es UML?

- UML = Unified Modeling Language
- Un lenguaje de propósito general para el modelado orientado a objetos. Impulsado por el Object Management Group (OMG, www.omg.org)
- UML combina notaciones provenientes desde:
  - Modelado Orientado a Objetos
  - Modelado de Datos
  - Modelado de Componentes
  - Modelado de Flujos de Trabajo (Workflows)

CONCEPTOS DE POO EN C#

- Espacio de nombres
- Clases
- Objetos
- Atributos
- Métodos
- Propiedades
- Encapsulación
- Herencia
- Polimorfismo
- Sobrecarga
- Abstracción
- Encapsulamiento
- Mensajes

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;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Ejemplo
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```


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*.

CLASES

- Los miembros de una clase pueden ser:
- **Métodos**: *Acciones que puede realizar*
- **Propiedades**: *Características que se pueden leer y, a veces, modificar*
- **Eventos**: *Notificaciones (eventos) que produce cuando se realiza una acción concreta*
- **Atributos**: *Variables internas y constantes*

CLASES EN UML

- Cada clase se representa en un rectángulo con tres compartimentos:
  - nombre de la clase
  - atributos de la clase
  - operaciones de la clase

OBJETOS

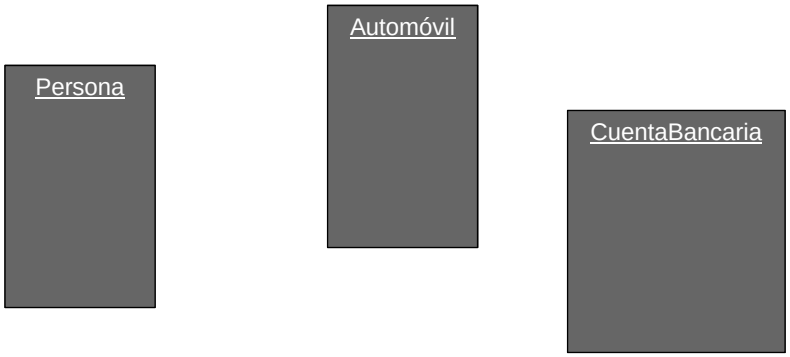
- Al crear un objeto se crea una instancia de la clase

```
Clase Objeto;
Objeto = new Clase( );

Clase Objeto = new Clase( );
```


OBJETOS EN UML

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



DECLARACIÓN DE CLASES

class <nombreClase>
{
<miembros>
}

- Para acceder a un campo de un determinado objeto se usa la sintaxis:
  - <objeto>**.**<campo>

DECLARACIÓN DE MÉTODOS

- La sintaxis que se usa en C# para definir los métodos es la siguiente:

<tipoDevuelto> <nombreMétodo> **(**<*parametros*>**)**
{
<instrucciones>
}

```
class Persona
{
    string Nombre;
        // Campo de cada objeto que almacena su
nombre

    int Edad;
        // Campo de cada objeto que almacena su edad

    public void Cumpleaños()  // Incrementa la edad del
objeto
    {
        Edad++;
    }
```


INVOCANDO MÉTODOS

```
class Arbol
{
    int Altura;
     public void Podar( )
     {
          Console.WriteLine("Podando …");
     }
}

Arbol Pino = new Arbol(); // Se crea el objeto
Pino.Podar();  //Se invoca el método Podar()
```

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

MIEMBROS ESTÁTICOS Y DE INSTANCIA (cont.)

- 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");
}
```

Miembros Estáticos

HERENCIA

- Es un mecanismo que permite definir nuevas clases a partir de otras ya definidas

class <nombreHija>**:**<nombrePadre>

{

<miembrosHija>

}

HERENCIA

- Modificadores:
  - *public*: De acceso desde cualquier parte del programa
  - *private*: De acceso exclusivo para los miembros de la clase que lo contiene
  - *protected*: De acceso solo a los miembros de la misma clase o clases derivadas. No se accesan desde el exterior.
  - *internal*: De acceso desde el mismo ensamblado, la misma clase y métodos de sus clases derivadas

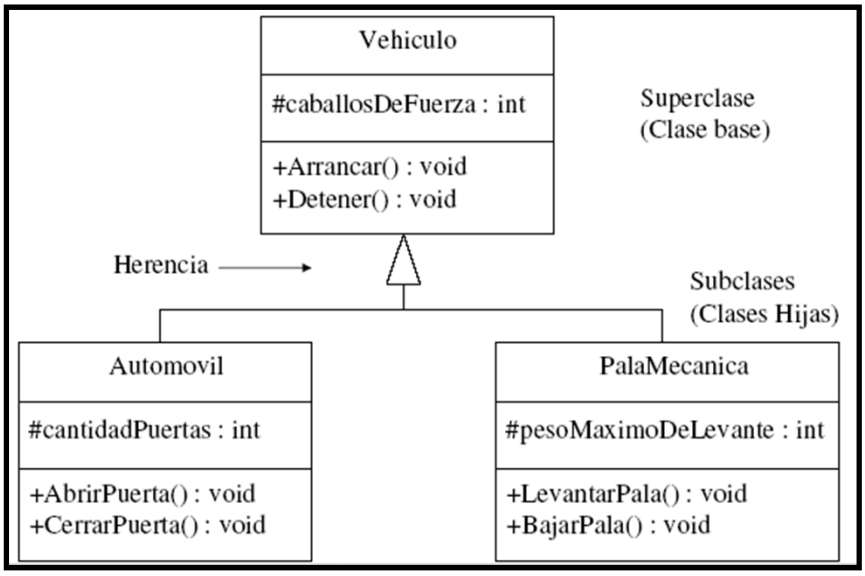
MODIFICADORES DE ACCESO

	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


```
class Clase
{
        int X; // Por default es "private"

        public int A;
        private char B;
        protected float C=5.67;
        internal double D;
        protected internal int E = 5;
}
```

HERENCIA



ENCAPSULACIÓN EN UML

- Los niveles de encapsulación están heredados de los niveles de C++:
  - **(-) Privado** : es el más fuerte. Esta parte es totalmente invisible (excepto para clases *friends* en terminología C++)
  - **(#)** Los atributos/operaciones **protegidos** están visibles para las clases *friends* y para las clases derivadas de la original
  - **(+)** Los atributos/operaciones **públicos** son visibles a otras clases (cuando se trata de atributos se está transgrediendo el principio de encapsulación)

POLIMORFISMO

- Es la capacidad de almacenar objetos de un determinado tipo en variables de tipos antecesores del primero a costa, claro está, de sólo poderse acceder a través de dicha variable a los miembros comunes a ambos tipos. Sin embargo, las versiones de los métodos virtuales a las que se llamaría a través de esas variables no serían las definidas como miembros del tipo de dichas variables, sino las definidas en el verdadero tipo de los objetos que almacenan.

LECTURAS ADICIONALES

Capítulo	Tema	Libro	Autor	Págs.
	Programación Orientada a Objetos en C# .NET (POO_Con_C_Sharp.PDF)	Filminas	Ing. Ramón Roque Hernández	

Se recomiendan estas lecturas para dominar los conceptos de Programación Orientada a Objetos