

# Programación II

## C #

### Clase 2

Maximiliano Neiner

Modificado por: Federico Dávila

# Temas a Tratar

- ❖ Programación Orientada a Objetos (POO)
- ❖ Clases
- ❖ NameSpaces

# Temas a Tratar

- ❖ Programación Orientada a Objetos (POO)
  - Características
  - Pilares
- ❖ Clases
- ❖ NameSpaces

# POO - ¿Qué es?

- ❖ Es una manera de construir Software basada en un nuevo paradigma.
- ❖ Propone resolver problemas de la realidad a través de identificar objetos y relaciones de colaboración entre ellos.
- ❖ El *Objeto* y el *Mensaje* son sus elementos fundamentales.

# Temas a Tratar

- ❖ Programación Orientada a Objetos (POO)
  - Características
  - Pilares
- ❖ Clases
- ❖ NameSpaces

# Pilares de POO



(1) Imagen tomada de <http://toodaim.blogspot.com.ar/2013/01/articulo-los-pilares-de-la-programacion.html>

# Abstracción

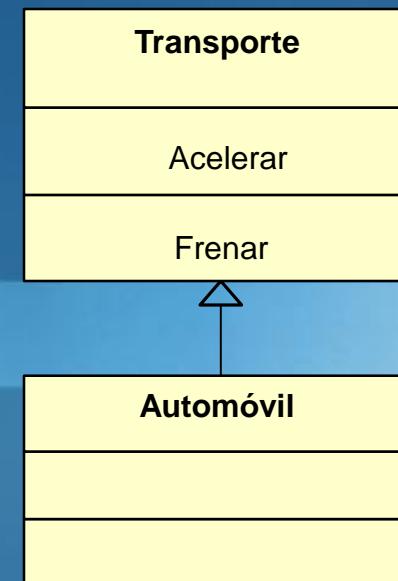
- ✖ Ignorancia selectiva.
- ✖ Decide qué es importante y qué no lo es.
- ✖ Se enfoca en lo que es importante.
- ✖ Ignora lo que no es importante.
- ✖ Utiliza la encapsulación para reforzar la abstracción.

# Encapsulamiento

- ❖ Esta característica es la que denota la capacidad del objeto de responder a peticiones a través de sus **métodos o propiedades** sin la necesidad de exponer los medios utilizados para llegar a brindar estos resultados.
- ❖ El exterior de la clase lo ve como una caja negra.

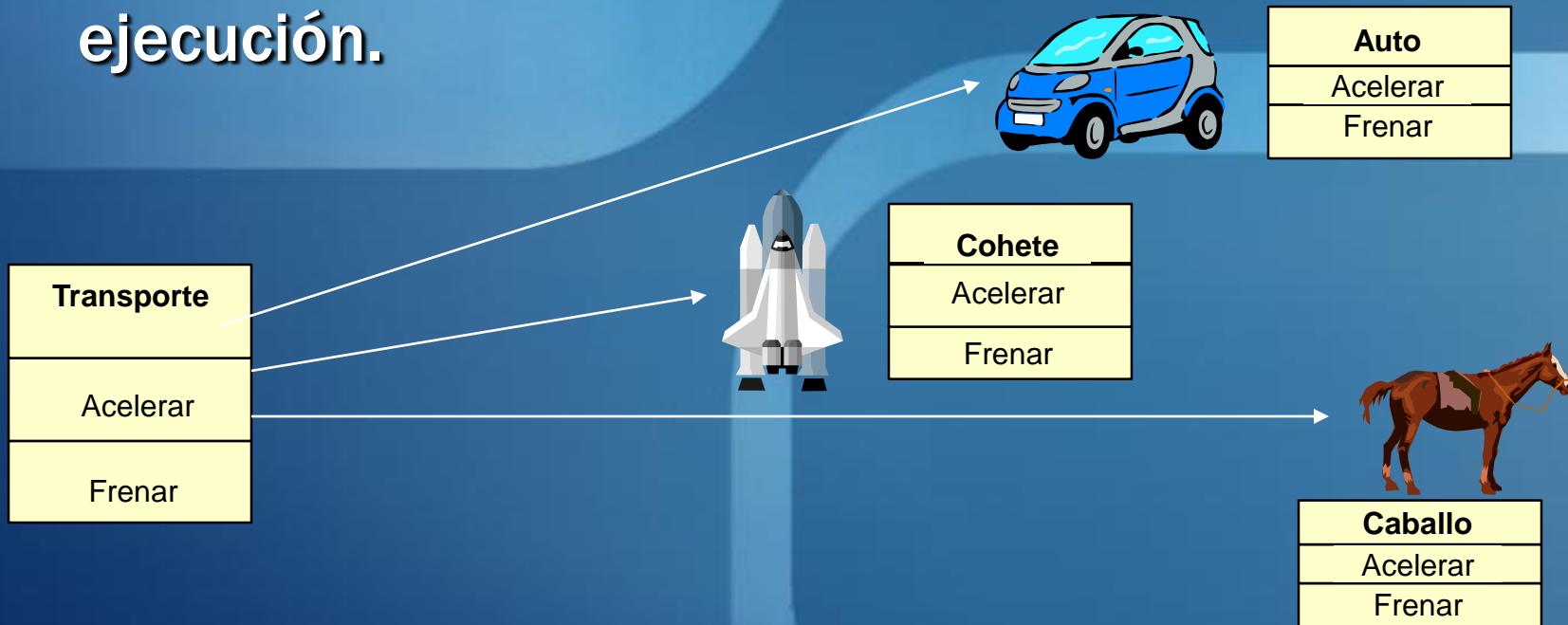
# Herencia

- ❖ Es “un tipo de” relación entre clases.
  - Relación “es un”
- ❖ Va de la generalización a la especialización.
- ❖ Clase base o padre.
- ❖ Clase derivada o hija.
- ❖ Hereda la implementación.



# Polimorfismo

- La definición del método reside en la clase base o padre.
- La implementación del método reside en la clase derivada o hija.
- La invocación es resuelta al momento de la ejecución.



# Temas a Tratar

❖ Programación Orientada a Objetos (POO)

❖ Clases

- Características
- Sintaxis
- Atributos
- Métodos

❖ NameSpaces

# ¿Qué es una clase?

- ❖ Una clase es una Clasificación.
- ❖ Clasificamos en base a comportamientos y atributos comunes.
- ❖ A partir de la clasificación se crea un vocabulario.
- ❖ Es una abstracción de un objeto.

# ¿Qué es una clase?

- ✖ Es una construcción **Estática** que describe:
  - Comportamiento común.
  - Atributos (estado).
- ✖ Estructura de datos.
- ✖ Incluye:
  - Datos
  - Métodos (definen comportamiento)



# Temas a Tratar

- ❖ Programación Orientada a Objetos (POO)
- ❖ Clases
  - Características
  - Sintaxis
  - Atributos
  - Métodos
- ❖ NameSpaces

# Sintaxis

```
[modificador] class Identificador
```

```
{  
    // miembros: atributos y métodos  
}
```

- ❖ **modificador:** Determina la accesibilidad que tendrán sobre ella otras clases.
  - ❖ **class:** Es una palabra reservada que le indica al compilador que el siguiente código es una clase.
  - ❖ **Identificador:** Indica el nombre de la clase.
    - Los nombres deben ser sustantivos, con la primera letra en mayúscula y el resto en minúscula.
    - Si el nombre es compuesto, las primeras letras de cada palabra en mayúsculas, las demás en minúsculas.
- Ejemplo: MiClase

# Modificadores

Nombre	Descripción
abstract	Indica que la clase no podrá instanciarse.
internal (*)	Accesible en todo el proyecto (Assembly).
public (*)	Accesible desde cualquier proyecto.
private (*)	Accesor por defecto.
sealed	Indica que la clase no podrá heredar.

(\*): Modificadores de visibilidad.

# Temas a Tratar

- ❖ Programación Orientada a Objetos (POO)
- ❖ Clases
  - Características
  - Sintaxis
  - Atributos
  - Métodos
- ❖ NameSpaces

# Sintaxis

[modificador] tipo identificador; // Igual que en C

- ❖ **modificador:** Determina la accesibilidad que tendrán sobre él las demás clases. Por defecto son **private**.
- ❖ **tipo:** Representa al tipo de dato. Ejemplo: int, float, etc.
- ❖ **Identificador:** Indica el nombre del atributo.
  - Los nombres deben tener todas sus letras en minúsculas.
  - Si el nombre es compuesto, la primera letra de la segunda palabra estará en mayúsculas, las demás en minúsculas.

**Ejemplo:**

```
string miNombre;
```

# Modificadores

Nombre	Puede ser accedido por...
private (*)	Los miembros de la misma clase.
protected	Los miembros de la misma clase y clases derivadas o hijas.
internal	Los miembros del mismo proyecto.
internal protected	Los miembros del mismo proyecto o clases derivadas.
public	Cualquier miembro. Accesibilidad abierta.

(\*): Accesor por defecto.

# Temas a Tratar

- ❖ Programación Orientada a Objetos (POO)
- ❖ Clases
  - Características
  - Sintaxis
  - Atributos
  - Métodos
- ❖ NameSpaces

# Sintaxis (Firma del método) (1/2)

```
[modificador] retorno Identificador ( [args] )
```

```
{  
    // Sentencias  
}
```

- ❖ **modificador:** Determina la forma en que los métodos serán usados.
  - ❖ **retorno:** Es el tipo de valor devuelto por el método (sólo retornan un único valor).
  - ❖ **Identificador:** Indica el nombre del método.
    - Los nombres deben ser verbos, con la primera letra en mayúscula y el resto en minúscula.
    - Si el nombre es compuesto, las primeras letras de cada palabra en mayúsculas, las demás en minúsculas.
- Ejemplo: AgregarAlumno

# Sintaxis (Firma del método) (2/2)

- ❖ **args:** Representan una lista de variables cuyos valores son pasados al método para ser usados por este. Los corchetes indican que los parámetros son opcionales.
- ❖ Los parámetros se definen como:

**Tipo\_Dato identificador\_parametro**

- ❖ Si hay más de un parámetro, serán separados por una coma ( , ).
- ❖ Si un método no retorna ningún valor se usará la palabra reservada **void**.
- ❖ Para retornar algún valor del método se utilizará la palabra reservada **return**.

# Modificadores

Nombre	Descripción
abstract	Sólo la firma del método, sin implementar.
extern	Firma del método (para métodos externos).
internal (*)	Accesible desde el mismo proyecto.
override	Reemplaza la implementación del mismo método declarado como <i>virtual</i> en una clase padre.
public (*)	Accesible desde cualquier proyecto.
private (*)	Sólo accesible desde la clase.
protected (*)	Sólo accesible desde la clase o derivadas.
static	Indica que es un método de clase.
virtual	Permite definir métodos, con su implementación, que podrán ser sobrescritos en clases derivadas.

# Ejemplo de una Clase

```
public class Automovil
{
    // Atributos NO estáticos
    public Single velocidadActual;
    // Atributos estáticos
    public static Byte cantidadRuedas;
    // Métodos estáticos
    public static void MostrarCantidadRuedas()
    {
        Console.WriteLine(Automovil.cantidadRuedas);
    }
    // Métodos NO estáticos
    public void Acelerar (Single velocidad)
    {
        this.velocidadActual += velocidad;
    }
}
```

# Temas a Tratar

- ❖ Programación Orientada a Objetos (POO)
- ❖ Clases
- ❖ NameSpaces
  - Características
  - Directivas
  - Creación
  - Miembros

# ¿Qué es un NameSpace?

- ❖ Es una agrupación lógica de clases y otros elementos.
- ❖ Toda clase está dentro de un NameSpace.
- ❖ Proporcionan un marco de trabajo jerárquico sobre el cuál se construye y organiza todo el código.
- ❖ Su función principal es la organización del código para reducir los conflictos entre nombres.
- ❖ Esto hace posible utilizar en un mismo programa componentes de distinta procedencia.

# Ejemplo de un NameSpace

❖ **System.Console.WriteLine()**

❖ Dónde:

- **System** es el NameSpace de la BCL (Base Class Library).
- **Console** es una clase dentro del NameSpace **System**.
- **WriteLine** es uno de los métodos de la clase **Console**.

# Temas a Tratar

- ❖ Programación Orientada a Objetos (POO)
- ❖ Clases
- ❖ NameSpaces
  - Características
  - Directivas
  - Creación
  - Miembros

# Directivas de un NameSpace

- ❖ Son elementos que permiten a un programa identificar los NameSpaces que se usarán en el mismo.
- ❖ Permiten el uso de los miembros de un NameSpace sin tener que especificar un nombre completamente cualificado.
- ❖ C# posee dos directivas de NameSpace:
  - Using
  - Alias

# Directiva Using

- ❖ Permite la especificación de una llamada a un método sin el uso obligatorio de un nombre completamente cualificado.

```
using System; //Directiva USING

public class Program
{
    public static void Main()
    {
        Console.WriteLine("Hola");
    }
}
```

# Directiva Alias

- ❖ Permite que un programa utilice un nombre distinto para un NameSpace.
- ❖ Es una técnica usada para conseguir una notación abreviada que evite el uso de NameSpaces largos.

```
using SC = System.Console;           //Directiva ALIAS

public class Program
{
    public static void Main()
    {
        SC.WriteLine("Hola, de nuevo");
    }
}
```

# Temas a Tratar

- ❖ Programación Orientada a Objetos (POO)
- ❖ Clases
- ❖ NameSpaces
  - Características
  - Directivas
  - Creación
  - Miembros

# Sintaxis

```
namespace Identificador
```

```
{  
    // Miembros  
}
```

- ☞ Dónde el identificador representa el nombre del NameSpace.
- ☞ Dicho nombre respeta la misma convención que las clases.

# Temas a Tratar

- ❖ Programación Orientada a Objetos (POO)
- ❖ Clases
- ❖ NameSpaces
  - Características
  - Directivas
  - Creación
  - Miembros

# Miembros de un NameSpace

Pueden contener ...

Clases

Delegados

Enumeraciones

Interfaces

Estructuras

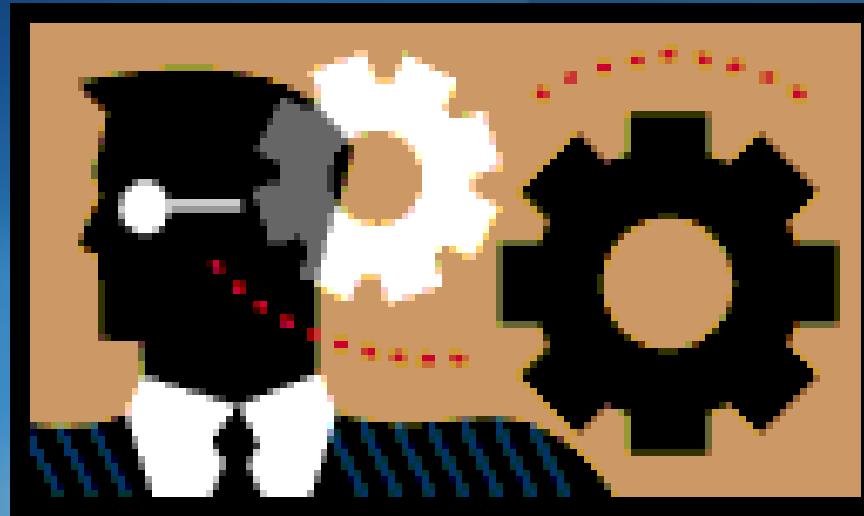
Namespaces

Directivas using

Directivas Alias

# Ejemplo de uso de NameSpaces

```
namespace MiNameSpace {  
    class MiClase {  
        public static int variable = 100;  
    }  
}  
  
namespace OtroNameSpace {  
  
    using SC = System.Console; //Directiva ALIAS  
  
    class MiClase {  
        int static variable = 1;  
  
        public static void Mostrar() {  
            SC.WriteLine("Mi variable {0}", MiClase.variable);  
            SC.WriteLine("La otra variable {0}", MiNameSpace.MiClase.variable);  
        }  
    }  
}
```



# Ejercitación