

ALGORITMOS Y PROGRAMACIÓN

Clase 4

Programación Orientada a Objetos

Año 2021 – 1er. Cuatrimestre

Temario

- POO
 - Características
- Clases
 - Características
 - Métodos
 - Constructores
 - Instanciación

Programación Orientada a Objetos

La Programación Orientada a Objetos es un paradigma de programación.

Cada paradigma:

- * provee un conjunto de patrones conceptuales que conducen el proceso de diseño y desarrollo de un programa.

- * brinda un conjunto de herramientas diferentes para comprender, analizar y representar problemas y expresar la solución en términos de una computadora.

Programación Orientada a Objetos

En POO se programa por '**simulación**', se '**personifican**' *los objetos físicos*, dándoles las características y la funcionalidad que ellos tienen en el mundo real.

Decimos entonces que un programa en este paradigma:

es un conjunto de OBJETOS que interactúan entre sí enviándose mensajes a través de los cuales solicitan la ejecución de una operación para llevar a cabo una tarea determinada.

¿Qué es un objeto?

- Uno puede mirar a su alrededor y ver muchos objetos del mundo real: un perro, un escritorio, un televisor, una bicicleta etc. Los sustantivos son un buen punto de partida para determinar los objetos de un sistema.
- Cada uno de los objetos presentan dos características:
 - Estado
 - Comportamiento



Un perro es un objeto que tiene

estado: nombre, color, raza.

comportamiento: ladrar, correr, jugar, etc.

Estado y comportamiento de un objeto

Estado

Describe las características, cualidades, atributos o propiedades de un objeto.

Un alumno tiene nombre y apellido, dni y legajo.

Un producto tiene nombre, código, marca y precio.

Comportamiento

Define el conjunto de operaciones que puede llevar a cabo un objeto (su funcionalidad)

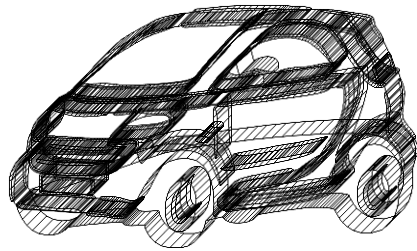
Se puede consultar el nombre y apellido de un alumno, ver su dni, modificar su legajo, etc.

Se puede guardar el nombre, la marca, el código y el precio de un producto, modificar su precio, etc..

¿Qué es un objeto?

- Formalmente un objeto es *una instancia* de una clase.
- Todas las instancias de una misma clase tienen los mismos atributos y el mismo comportamiento.
- Pero cada instancia **tiene sus propios valores** para cada estado

Estado de un auto:
color, marca, tamaño,
precio



Color
verde
Tamaño
pequeño



Color
amarillo
Tamaño
mediano



Color rojo
Tamaño
grande

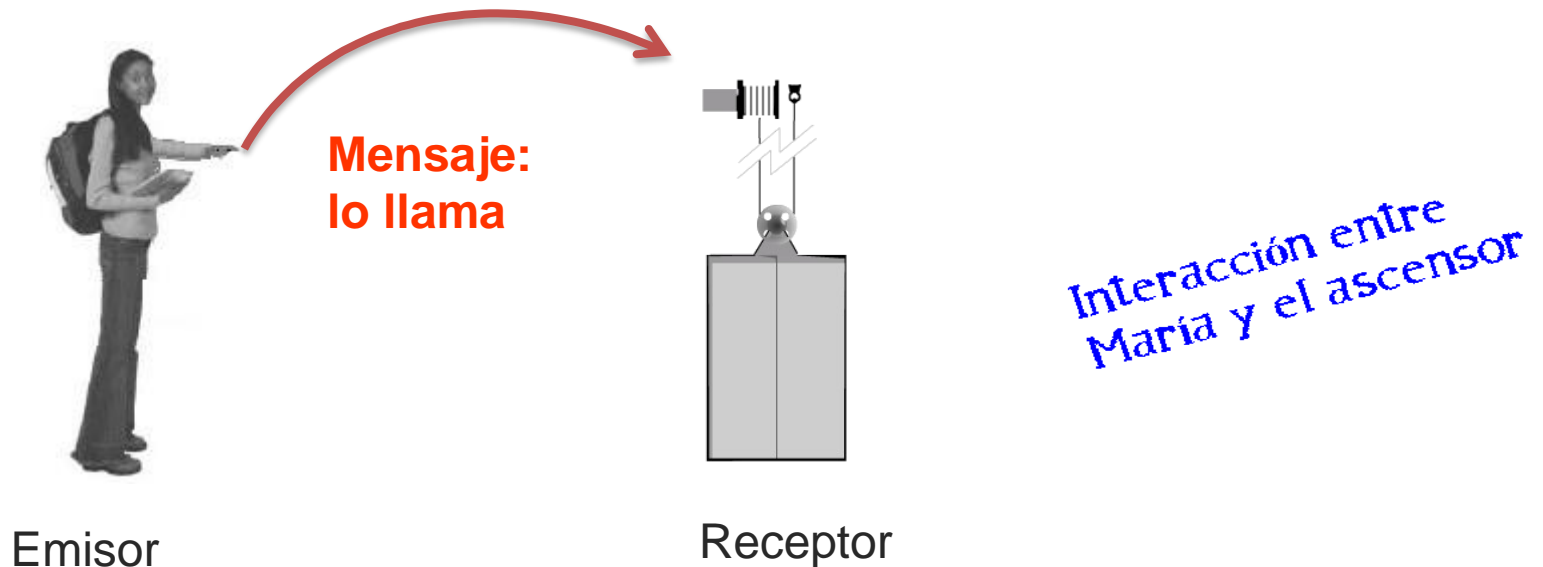


Cómo interactúan los objetos?

Supongamos que una persona, María, está en PB y quiere tomar el ascensor para llegar a su departamento.

En este problema **María es un objeto** y el **ascensor es otro objeto**.

María llama al ascensor.



Cómo interactúan los objetos?

- El ascensor llega y María entra. María le indica a que piso quiere ir. El ascensor cierra la puerta y la lleva hasta el piso seleccionado.

En esta secuencia podemos observar la interacción existente entre los dos objetos que hemos identificado:

María se comunicó con el ascensor → a través de un **mensaje** lo llamó

El ascensor tiene la **responsabilidad** de ir al lugar donde María lo llamó. → **Responde** yendo al piso donde está María.

María sube al ascensor y a través de un nuevo **mensaje** le dice que la lleve a un determinado piso.

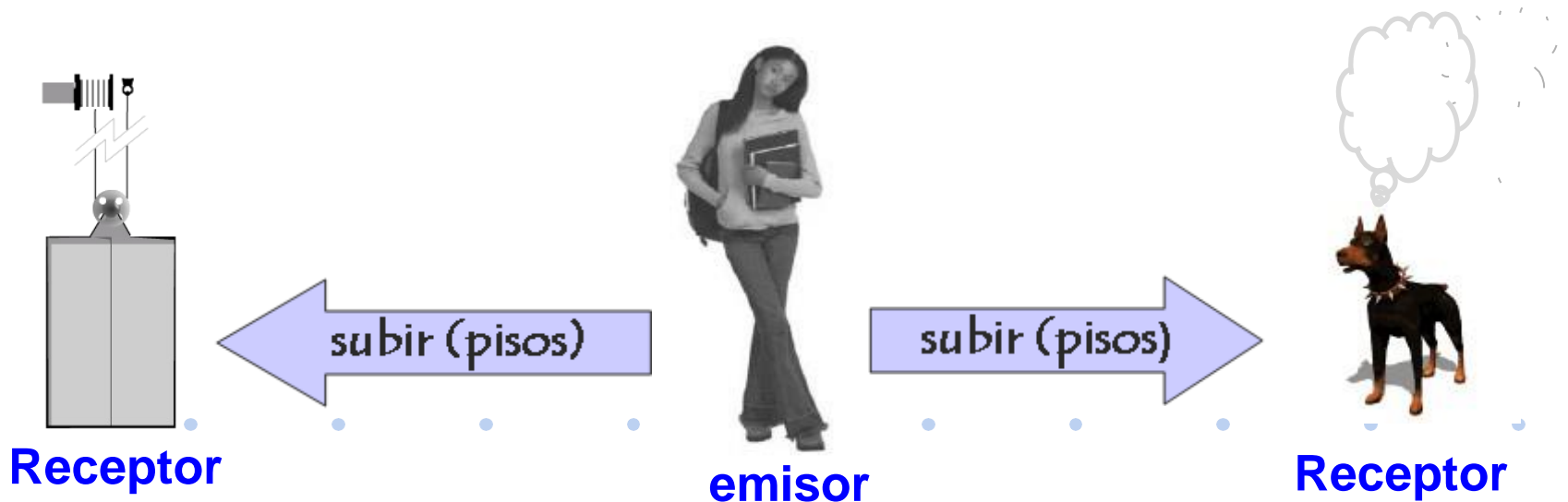
El ascensor tiene la **responsabilidad** de llevarla al piso indicado. → **Responde** yendo al piso que le indicó María.

Cómo interactúan los objetos?

Los objetos interactúan enviándose *mensajes* y responden ejecutando *un método* que contiene las acciones correspondientes al mensaje recibido.

En un mensaje, siempre hay **un emisor** que es el objeto que lo envía y **un receptor que** es el objeto que lo recibe.

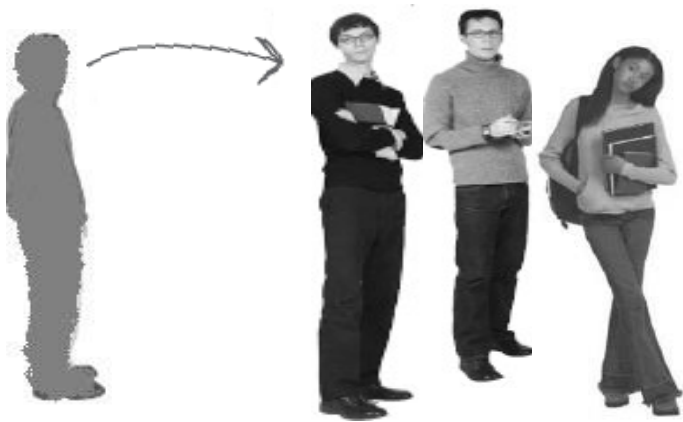
La interpretación del mensaje, es decir, el método usado para responder al mensaje es determinado por el receptor y podría variar dependiendo de quién lo recibe.



Clases e instancias.

María, al igual que otras personas que deciden tomar el ascensor, tienen características comunes, esto es porque pertenecen a una **Categoría** o **Clase** que podríamos llamar "**pasajero**".

María y otras personas que usan el ascensor son **instancias** de la **clase** pasajero



Una **clase** es un molde a partir de la cual se **crean instancias** con las mismas características y comportamiento.

¿Qué es una clase?

- Una *clase* es un molde o plantilla que permite crear objetos o instancias de dicha clase.
- Todo objeto pertenece a una clase (Clasificación)
- Se hace la clasificación en base a comportamiento y atributos comunes.
- Una clase es el repositorio de los atributos y el comportamiento común de todos los objetos que pertenecen a dicha clase.



Abstracción: base de la POO

- Se *abstraen* de los objetos del mundo real sus *características esenciales y su funcionalidad* y se los *encapsula y oculta* en una clase.
- Una clase se define a través de:
 - una *estructura de datos*, representada por las *variables de instancia*, que describe los atributos de los objetos (estado)
 - de un conjunto de *funciones*, llamadas *métodos*, que definen el comportamiento.
- El estado de un objeto sólo se puede cambiar mediante los métodos, o sea, a través de las operaciones definidas para el mismo.

Clase - Ejemplo

- ¿Cuál es el estado y el comportamiento que tienen en común todos los autos?



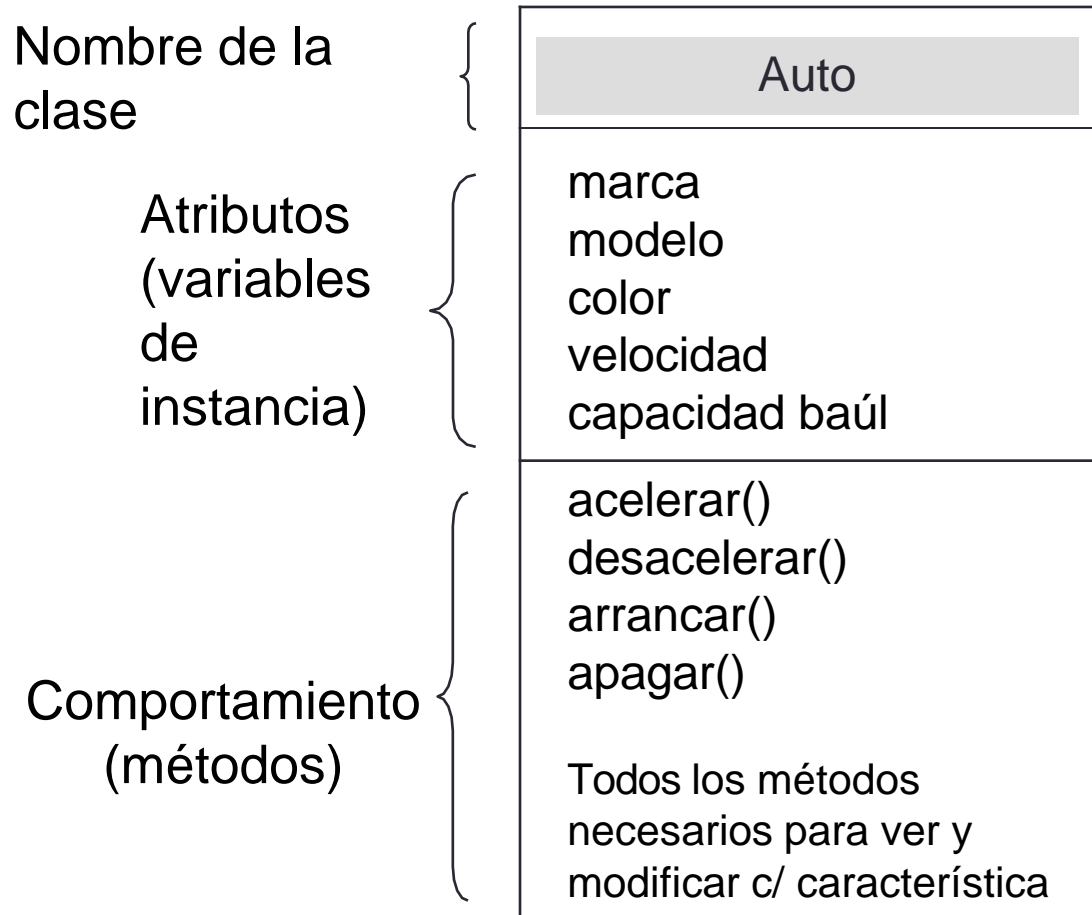
Modelo
Marca
Color
Velocidad

Atributos
(estado)

Acelerar
Desacelerar
Apagar
Arrancar

comportamiento

Clase de objetos



La clase encapsula el estado y el comportamiento de los objetos de dicha clase.

Clases en .Net


.NET está organizado en clases: la BCL es un repositorio de *clases* y toda la funcionalidad que nos provee .NET está implementadas en *clases*.

Program.cs

```
using System;
namespace Ejercicio1
{
    class Program
    {
        public static void Main(string[] args)
        {
            ...
        }

        public static void HacerAlgo()
        {
            ...
        }

        public static void NoHacerNada()
        {
            ....
        }
    }
}
```

A diagram with a grey rounded rectangle containing the text "Todo el código ejecutable tiene que estar dentro de funciones pertenecientes a alguna clase." Three red arrows point from this box to the code: one to the 'class Program' line, and two to the 'Main' and 'HacerAlgo()' method signatures.

Todo el código ejecutable tiene que estar dentro de funciones pertenecientes a alguna clase.

- Por defecto los proyectos utilizan la función Main de la clase Program como punto de entrada para la ejecución de cualquier programa.

Creación de Clases en C#

- Sintaxis de definición de clases

```
class <nombreClase>
{
    <definición de los datos miembros>
    <definición de los funciones miembros>
}
```

Donde:

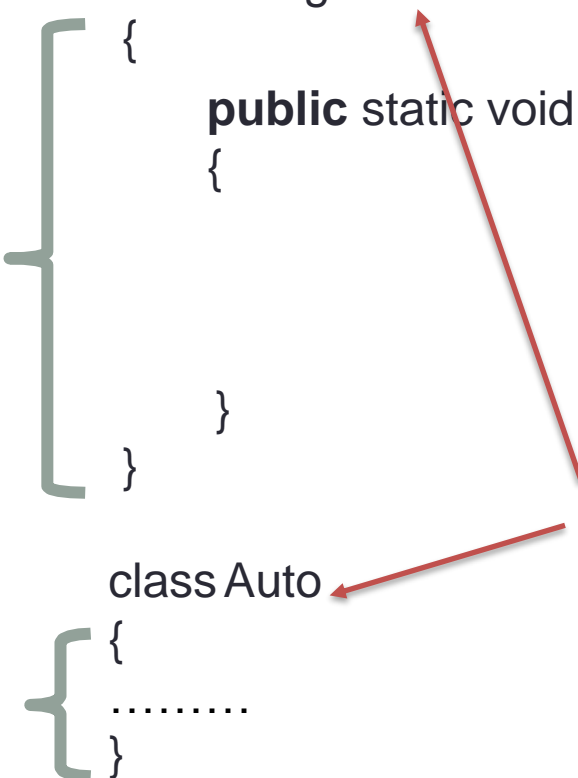
- **datos miembro**, son los atributos (variables de instancia)
- **funciones miembro**, son los métodos

Creación de Clases en C#

1ra forma: En este ejemplo la nueva clase está definida en el proyecto que la usa: se escribe en el mismo archivo debajo de la clase Program, dentro del mismo namespace.

```
namespace Ejercicio1
{
    class Program
    {
        public static void Main(string[] args)
        {
        }
    }

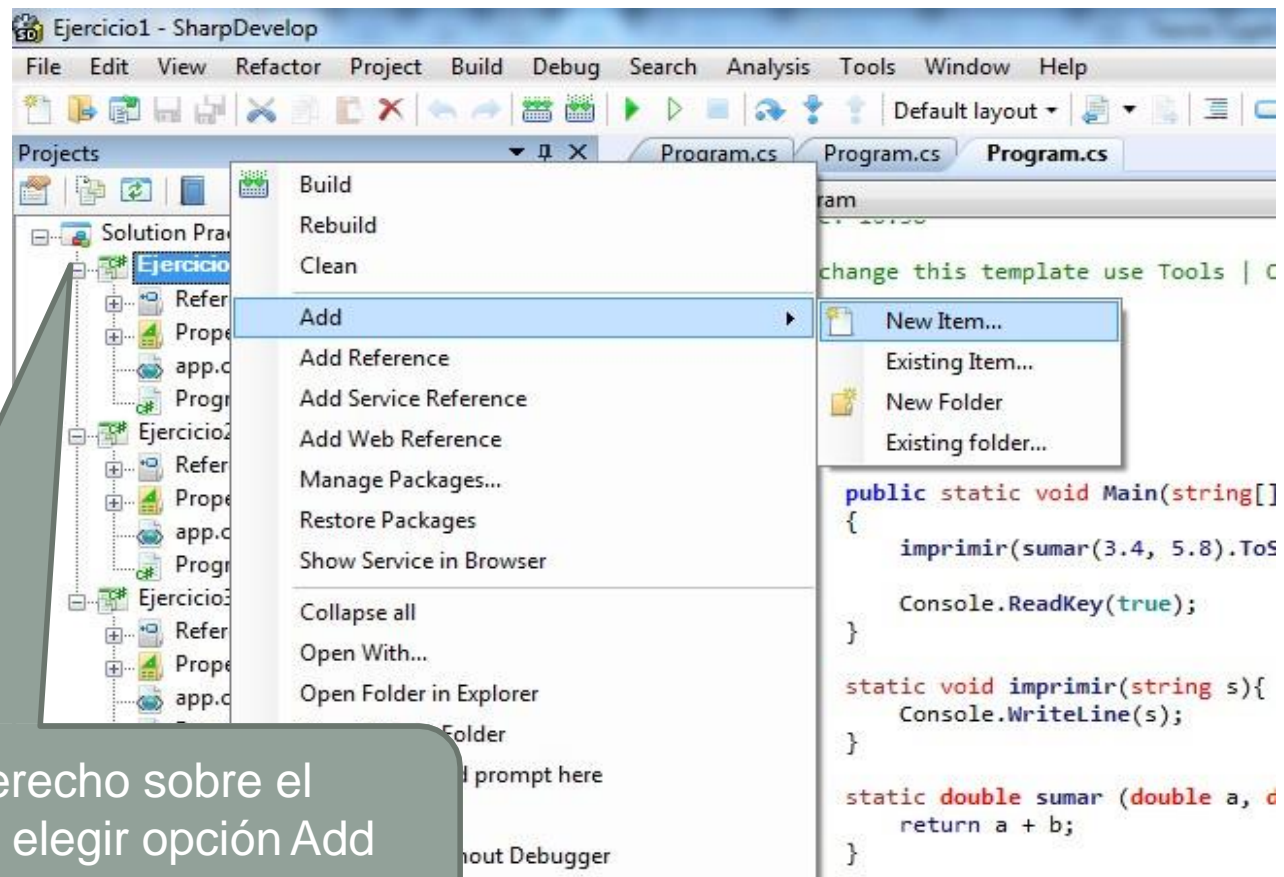
    class Auto
    {
        .....
    }
}
```



Recordar que cada clase debe tener su propio bloque de código encerrado entre llaves, dentro del mismo namespace.

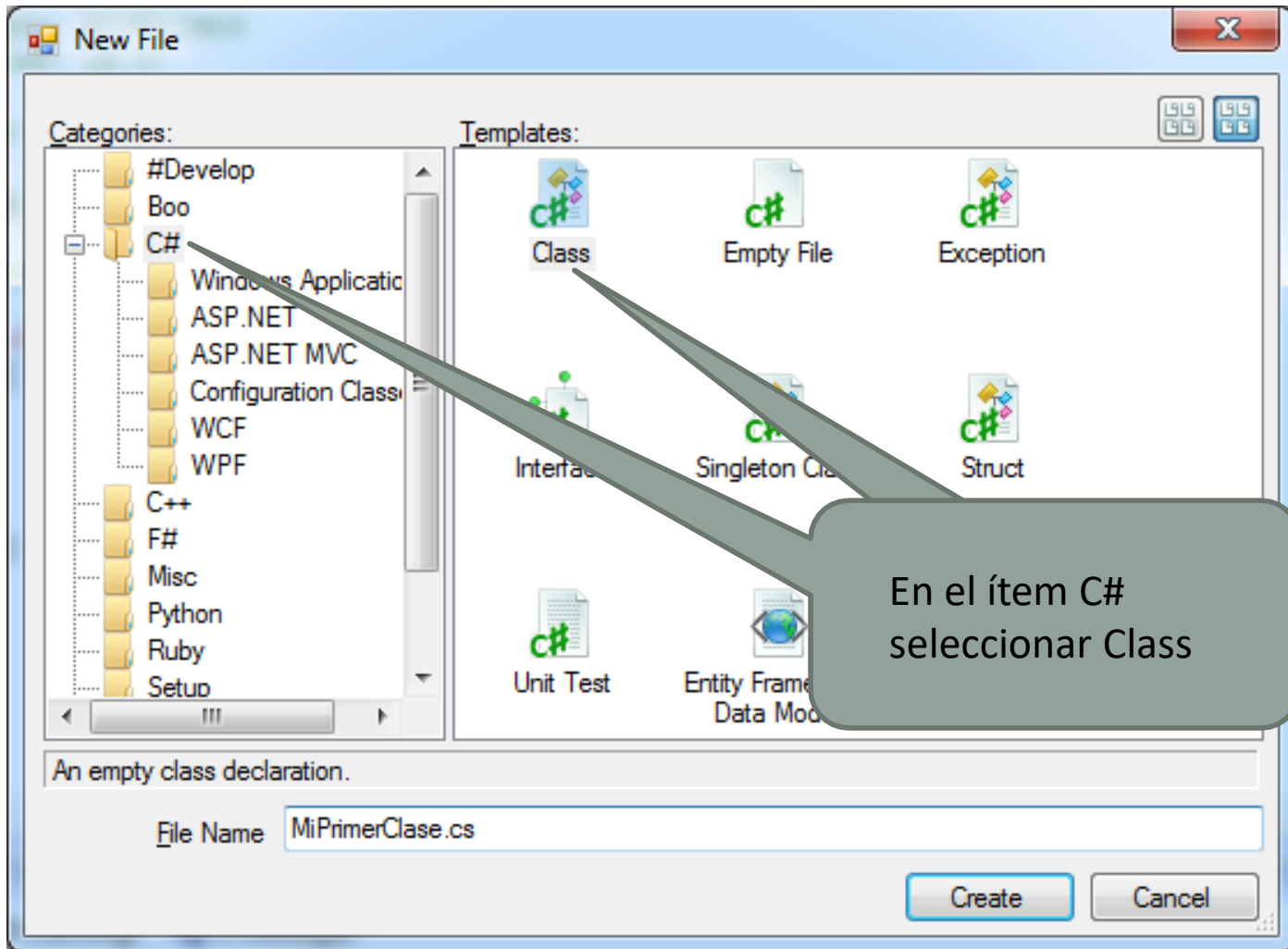
Creación de Clases en C#

2da forma: En este ejemplo la nueva clase estará definida en otro archivo que se agrega al proyecto, siempre dentro del mismo namespace.

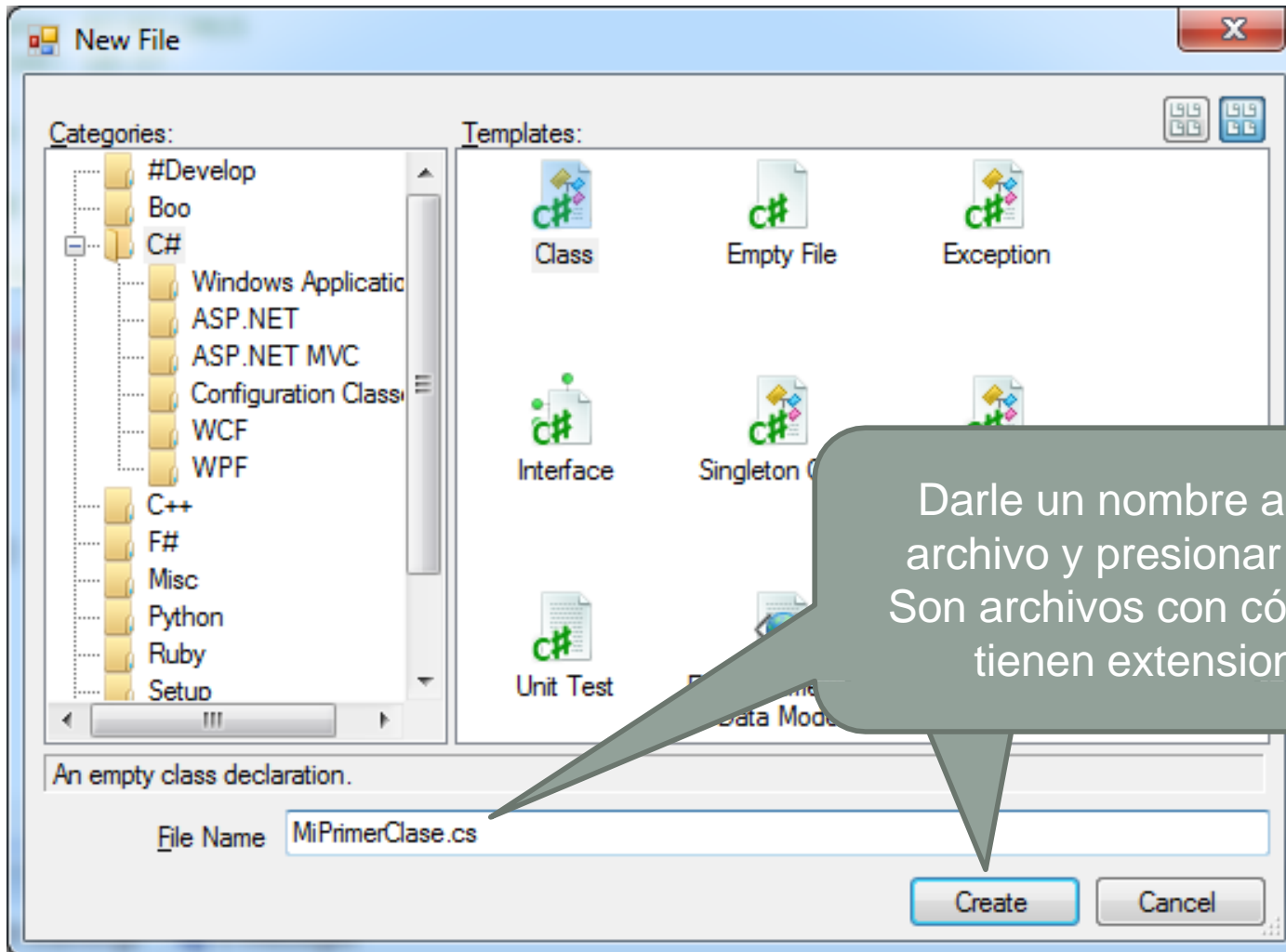


Click derecho sobre el proyecto y elegir opción Add
→ New Item...

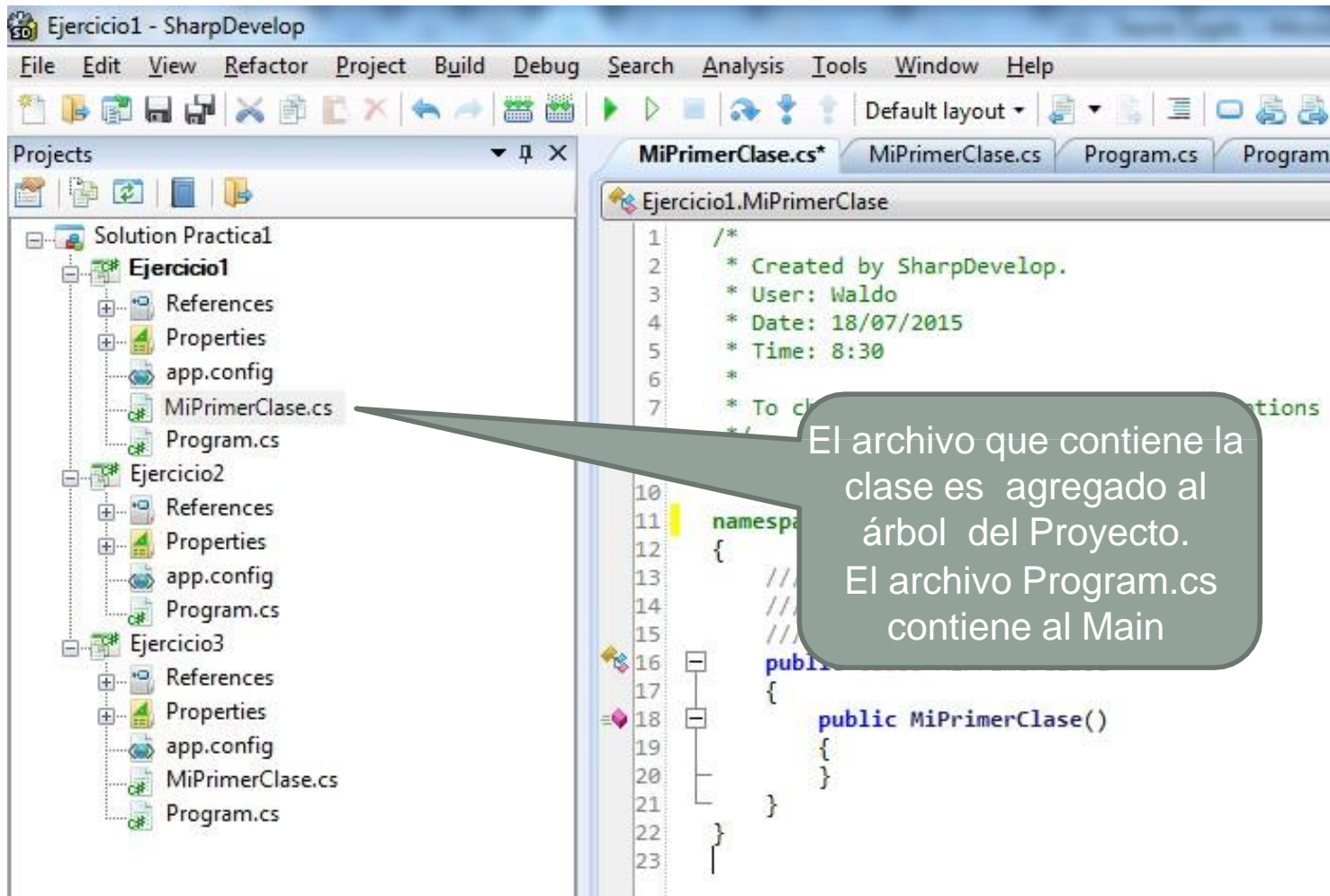
Agregando una nueva clase al proyecto



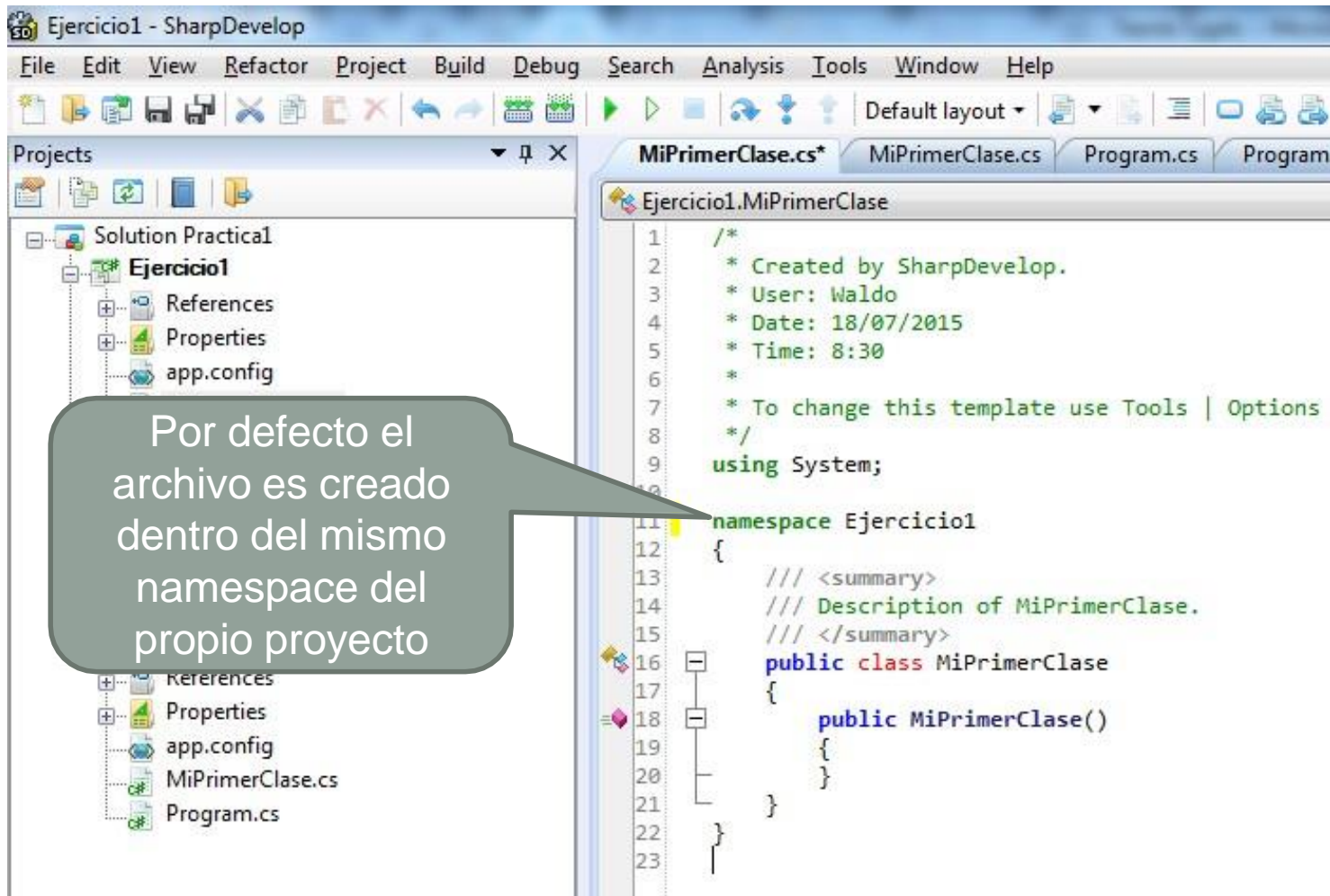
Agregando una nueva clase al proyecto



Agregando una nueva clase al proyecto



Agregando una nueva clase al proyecto



Creando una función dentro de la clase

```
namespace Ejercicio1  
{
```

```
    public class MiPrimerClase  
    {
```

```
        .....
```

```
        .....
```

```
        public void Imprimir()  
        {  
            Console.WriteLine("Perfecto, funcionó");  
        }
```

```
    }
```

```
}
```

Agrega esta
función o
método

En un archivo está la clase

MiPrimerClase.cs

Usando la nueva clase desde el Main

```
namespace Ejercicio1
{
    class Program
    {
        public static void Main(string[] args)
        {
            MiPrimerClase m1;
            m1=new MiPrimerClase();
            m1.Imprimir();

            Console.ReadKey(true);
        }
    }
}
```

Al estar la clase en el mismo namespace, el Sistema Operativo reconoce su uso.

MiPrimerClase m1;
m1=new MiPrimerClase();
m1.Imprimir();

Console.ReadKey(true);

En otro archivo está el Main

Program.cs

Clases en C# Instanciando objetos

- Una vez creada la clase se pueden **crear o instanciar** objetos de dicha clase.
- Para crear una instancia se utiliza el operador **new** y se debe especificar la clase del objeto a crear.

Si suponemos creada la clase Auto, hacemos:

a1=new Auto(); y creamos un objeto de dicha clase.

- Se pueden crear tantas instancias de una clase como se necesite.

Clases en C#

```
namespace Ejercicio1
```

```
{
```

```
    class Program
```

```
    {
```

```
        public static void Main(string[] args)
```

```
        {
```

```
            Auto a1;
```

```
            Auto a2;
```

```
            a1 = new Auto();
```

```
            a2 = new Auto();
```

```
        }
```

```
    }
```

```
    class Auto
```

```
    {
```

```
    }
```

```
}
```

Declaración de
dos variables
de tipo Auto:
a1 y a2

Creación de dos
instancias de la
clase Auto: una
almacenada en la
variable a1 y otra
en a2

Clases en C#

Campos/variables de instancia

- A los atributos o características que describen o identifican a los objetos de una determinada clase se los denomina ***campos o variables de instancia***.
- Se definen dentro de la clase con la siguiente sintaxis:

```
<tipoCampo> <nombreCampo>;
```

Clases en C#

Campos/variables de instancia

- Por ejemplo, en la clase **Auto** definimos dos campos : `marca` y `modelo`.

```
class Auto
{
    private string marca;
    private int modelo;
}
```

Por ahora cada vez que declaremos un campo le agregaremos el modificador **private**. Significa que solo pueden ser usados dentro de la clase.

Clases en C# - Métodos

- A las funciones u operaciones que se implementan en una clase se las llaman *métodos*.
- **Los métodos permiten manipular los datos almacenados en los objetos (modificarlos y consultarlos).**

Dentro de los métodos puede accederse a todos los campos de la clase.

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

```
<tipoResultado> <nombreMétodo> (<parametros>)  
{  
    <instrucciones>  
}
```

Clases en C# - Métodos

- Ejemplo: Definiendo el método `imprimir()` en la clase `Auto`, permite mostrar el valor de sus variables de instancia desde el código fuera de la clase.

```
class Auto {  
    private string marca;  
    private int modelo;  
    public void imprimir(){  
        Console.WriteLine("Marca " + marca + " modelo " + modelo);  
    }  
}
```

Por ahora cada vez que declaremos una función le agregaremos el modificador **public**. De esta forma permitimos que sea invocado en cualquier lado.

Clases en C# - Métodos

```
public static void Main(string[] args)
{
    Auto a1;
    a1 = new Auto();
    .....
    .....
    a1.imprimir();
    a2.imprimir();
}
```


Sobrecarga de métodos en C#

- La firma de un método consiste en:
 - El nombre
 - El número de parámetros
 - El tipo y el orden de los parámetros
 - Los modificadores de los parámetros

```
<tipoResultado> <nombreMétodo> (<parametros>)  
{  
    <instrucciones>  
}
```

- El tipo de resultado no es parte de la firma. Los nombres de los parámetros tampoco son parte de la firma.
- Una clase puede tener más de un método con el mismo nombre siempre que sus firmas sean diferentes. En este caso se produce la sobrecarga.

Sobrecarga de métodos

- Agreguemos un nuevo método "acelerar" con tres sobrecargas

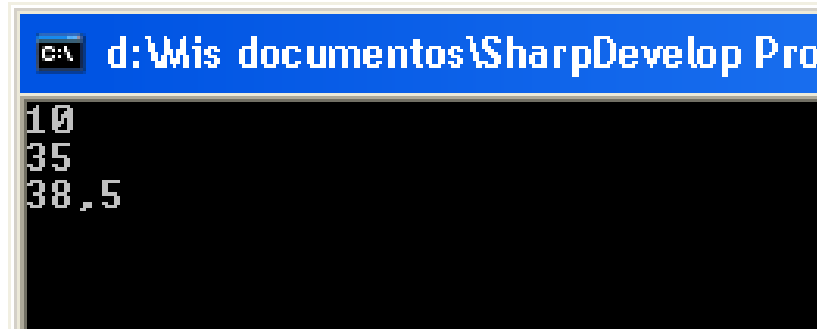
```
private double velocidad = 0;  
public double acelerar() {  
    return velocidad+= 10;  
}  
public double acelerar(int valor) {  
    return velocidad+= valor;  
}  
public double acelerar(double coeficiente) {  
    return velocidad*= coeficiente;  
}
```

Sobrecarga de métodos

```
public static void Main(string[] args)
{
    Auto a1 = new Auto();

    Console.WriteLine(a1.acelerar());
    Console.WriteLine(a1.acelerar(25));
    Console.WriteLine(a1.acelerar(1.1));
}
```

Cómo sabe
cuál método
acelerar
debe usar en
cada caso?
Por su firma!



The screenshot shows a Windows command prompt window with a blue title bar. The title bar text is "d:\Mis documentos\SharpDevelop Pro". The command prompt has a black background with white text. The output of the program is displayed on three lines: "10", "35", and "38,5".

```
d:\Mis documentos\SharpDevelop Pro
10
35
38,5
```

Constructores

- Un *constructor* definido en una clase es un **método especial** que contiene código a ejecutar cada vez que se crea una instancia de esa clase.
- La sintaxis de un constructor consiste en definirlo como cualquier otro método pero dándole el **mismo nombre que la clase** y no indicando el tipo de valor de retorno.

```
<modificadores> <nombreClase>(<parámetros>)  
{  
    <código>  
}
```

Constructores

- En la clase **Auto** definimos un constructor para poder asignar valor a las variables de instancia en el momento de crear el objeto.

```
class Auto {  
    private string marca;  
    private int modelo;  
    public Auto(string marca, int modelo) {  
        this.marca = marca;  
        this.modelo = modelo;  
    }  
    public void imprimir(){  
        Console.WriteLine("Marca y modelo: {0} {1}", marca, modelo);  
    }  
}
```

En el constructor utilizamos `this.marca` para diferenciar la variable de instancia `marca`, del parámetro `marca` del método. Idem con `modelo`.

Constructores

```
class Auto {  
    private string marca;  
    private int modelo;  
    public Auto(string mar, int mod) {  
        marca = mar;  
        modelo = mod;  
    }  
    public void imprimir(){  
        Console.WriteLine("Marca y modelo: {0} {1}", marca, modelo);  
    }  
}
```

Si los parámetros reciben otro nombre distinto al de las variables de instancia, entonces no se usa la palabra this.

Constructores

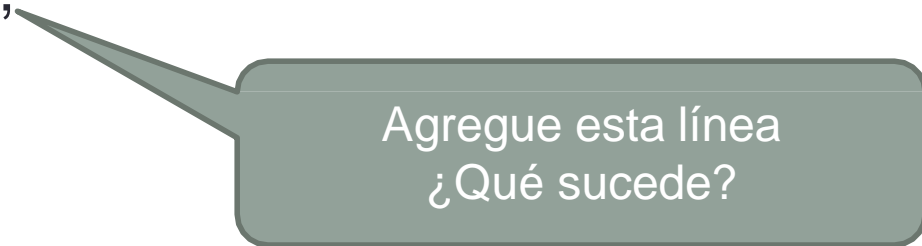
```
public static void Main(string[] args)
{
    Auto a1 = new Auto("Fiat", 2000);
    Auto a2 = new Auto("Ford", 2001);
    a1.imprimir();
    a2.imprimir();

    Console.ReadKey(true);
}
```

Constructores

```
public static void Main(string[] args)
{
    Auto a1 = new Auto("Fiat", 2000);
    Auto a2 = new Auto("Ford", 2001);
    Auto a3 = new Auto();
    a1.imprimir();
    a2.imprimir();

    Console.ReadKey(true);
}
```



Agregue esta línea
¿Qué sucede?

Constructores por defecto

En caso de no definir un constructor para la clase el compilador creará uno por defecto:

```
<nombreClase>()  
{  
}
```



```
public Auto()  
{  
}
```

Si nosotros definimos un constructor, el compilador **no incluye** ningún otro constructor.

Por ello **Auto a3=new Auto();** da error de compilación pues el constructor por defecto no existe más.

Constructores

Se puede definir más de un constructor, siempre que sus firmas sean diferentes.

```
class Auto
```

```
{
```

```
    private string marca;
```

```
    private int modelo;
```

```
    public Auto(){
```

```
}
```

Constructor que no recibe
parámetros

```
    public Auto(string marca, int modelo){
```

```
        this.marca = marca;
```

```
        this.modelo = modelo;
```

```
}
```

Constructor que recibe un
string y un int

```
    public Auto(string marca, string modelo){
```

```
        this.marca = marca;
```

```
        this.modelo = Int64.Parse(modelo);
```

```
}
```

Constructor que recibe dos
strings

```
    public void imprimir(){
```

```
        Console.WriteLine("Marca " + marca + " modelo " + modelo);
```

```
}
```

```
}
```

Constructores

```
class Auto
```

```
{
```

```
    private string marca; private
```

```
    public Auto(){
```

```
}
```

```
    public Auto(string marca
```

```
        this.marca = marca;
```

```
        this.modelo = modelo;
```

```
}
```

```
    public Auto(string marca, string modelo){
```

```
        this.marca = marca;
```

```
        this.modelo = int.Parse(modelo);
```

```
}
```

```
    public void imprimir(){Console.WriteLine("Marca " + marca + " " + modelo " + modelo);
```

```
}
```

```
}
```

```
Auto a2 = new Auto("Fiat", 2000);
```

```
Auto a3 = new Auto("Ford", "2010");
```

```
Auto a1 = new Auto();
```

Puesta en común Ejercicio 1 – TP 4

Codifique la clase Hora de tal forma que al ejecutar el siguiente programa de aplicación (Main) se imprima por consola:

23 HORAS, 30 MINUTOS Y 15 SEGUNDOS

```
class Program {  
    public static void Main(string[] args)  
    {  
        Hora h=new Hora(23,30,15);  
        h.imprimir();  
        Console.ReadKey(true);  
    }  
}
```

- Crear un nuevo archivo dentro del proyecto para representar la clase Hora.
- ¿Qué variables de instancia podemos definir en la clase Hora?

Miembros de una clase.

- En POO hay dos tipos de miembros (tanto para variables como para métodos)
 - De instancia
 - De clase
- En C# si queremos declarar *miembros de clase* usamos la palabra reservada *static*.

```
public static <tipo> unaVariableDeClase;  
public static void UnMetodoDeClase() { }
```

Si queremos que los *miembros sean de instancia* no ponemos nada

```
public <tipo> variableDeInstancia;  
public void UnMetodoDeInstancia() { }
```

Miembros de instancia

- Los miembros de instancia, ya sean variables o métodos son utilizados cuando se trabaja con **instancias u objetos**.

```
class Auto
```

```
{
```

```
    private string marca;
```

```
    private int modelo;
```

Variables de
instancia

```
    public Auto(){
```

```
    }
```

```
    public Auto(string marca, int modelo){
```

```
        this.marca = marca;
```

```
        this.modelo = modelo;
```

```
    }
```

```
    .....
```

```
    public void imprimir(){
```

```
        Console.WriteLine("Marca " + marca + " modelo " + modelo);
```

```
    }
```

```
}
```

Método de
instancia

Miembros de instancia

En el siguiente ejemplo creamos objetos o instancias a1, a2 y a3 de la clase Auto:

```
Auto a1 = new Auto("Ford", 1987);  
Auto a2 = new Auto("Peugeot", 1995);  
Auto a3 = new Auto("Fiat", 2008);
```

```
a1.imprimir();  
a2.imprimir();  
a3.imprimir();
```

Miembros de instancia

- Cuando se instancia una clase, el compilador genera en memoria una "copia" de la clase para cada instancia creada.

RAM

a1

marca = "Ford"
modelo = 1987

a2

marca = "Peugeot"
modelo = 1995

a3

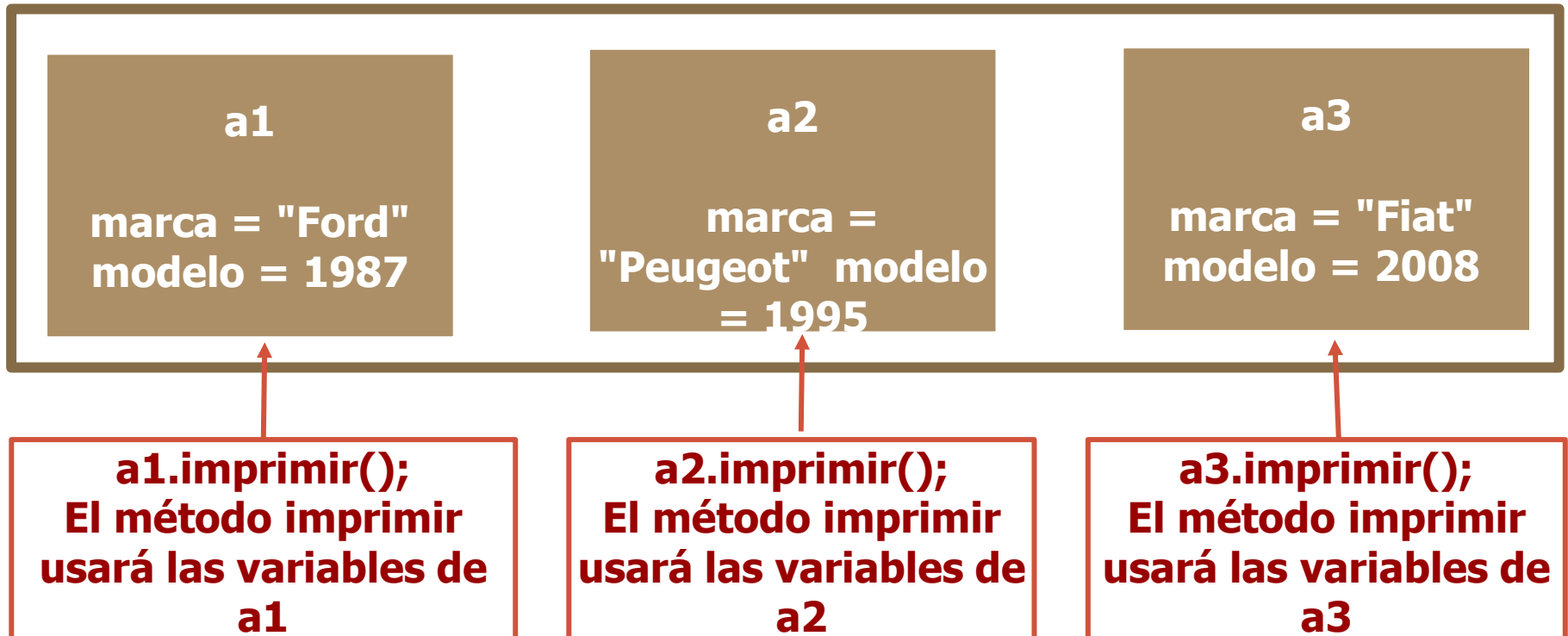
marca = "Fiat"
modelo = 2008

Marca y modelo son las variables de instancia de la clase Auto. Técnicamente hay tres variables marca y tres variables modelo

Miembros de instancia

Los métodos de instancia son los que acceden a cada variable dependiendo de quién sea el objeto receptor del mensaje.

RAM



Puesta en común Ejercicio 3 – TP 4

Defina una clase Persona con 3 campos: Nombre, Edad y DNI. Escriba un programa de aplicación (Main) que permita al usuario ingresar por consola una serie de datos de la forma

"Nombre<TAB>Documento<TAB>Edad<ENTER>".

El proceso de entrada finaliza con un string vacío.

Una vez finalizada la entrada de datos, el programa debe imprimir en la consola el listado con la forma:

Nro.) Nombre (Edad) <TAB> DNI.

Por ejemplo:

1) Juan Perez (40) 2098745

2) José García (41) 1965412

Vamos por partes...

Definamos una clase Persona con 3 campos: nombre, edad y dni

1- Qué tipos de miembros son nombre, edad y dni?

2- ¿Dónde definimos la operación imprimir? Es un método de

3- En la aplicación, que estructura de datos utilizamos para almacenar a las personas?

4- Cómo ingresamos los datos de entrada? (split)

Miembros de clase

- Los miembros de clase, ya sean variables de clase o métodos, no pertenecen a ningún objeto en particular, *son comunes y compartidos por todas las instancias u objetos* que pertenecen a la clase.
- Se declaran con la palabra reservada **static**
- La referencia a un miembro de clase se hace mediante el nombre de clase

<clase> . <miembro>

Miembros de Clase

```
class Auto {
```

```
    public string marca;
```

```
    public int modelo;
```

```
    public static int impresiones = 0;
```

Declaración de
una variable
de clase

```
    public Auto(string marca, int modelo){
```

```
        this.marca = marca;
```

```
        this.modelo = modelo;
```

```
    }
```

```
    public void imprimir(){
```

```
        Console.WriteLine("Marca y modelo: {0} {1}", marca, modelo);  
        impresiones++;
```

```
    }
```

```
}
```

La variable es usada como una variable más.
En este ejemplo la usamos para llevar la
contabilidad de la cantidad de veces que se
imprime un auto.

Miembros de clase

```
class Program
{
    public static void Main(string[] args)
    {
        Auto a1 = new Auto("Ford", 1987);
        Auto a2 = new Auto("Peugeot", 1999);
        Auto a3 = new Auto("Fiat", 2008);
        a1.imprimir();
        a2.imprimir();
        a3.imprimir();
        Console.WriteLine(Auto.impresiones);
    }
}
```

Notar que para hacer referencia a una variable de clase en una aplicación, se usa el nombre de la propia clase

Miembros de clase

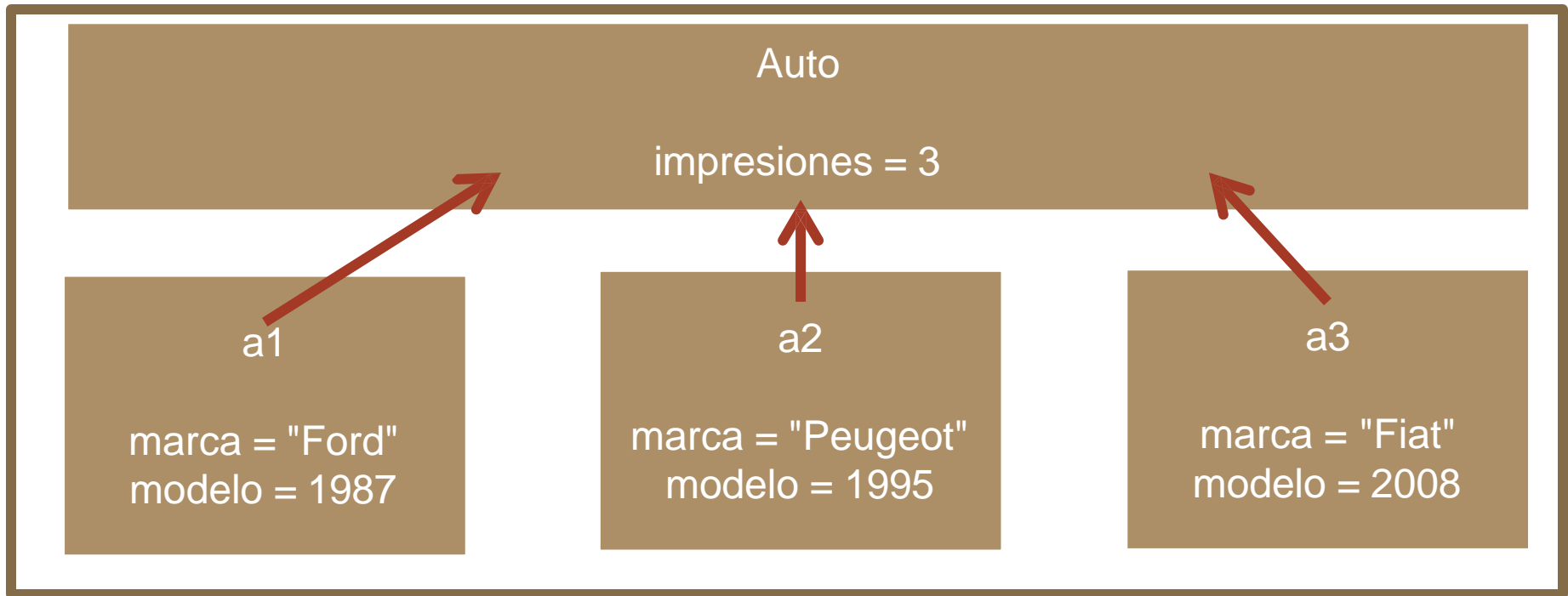
```
class Program
{
    public static void Main(string[] args)
    {
        Auto a1 = new Auto("Ford", 1987);
        Auto a2 = new Auto("Peugeot", 1995);
        Auto a3 = new Auto("Fiat", 2008);
        a1.imprimir();
        a2.imprimir();
        a3.imprimir();
        Console.WriteLine(Auto.impresiones);
    }
}
```



¿Qué imprime?

Miembros de clase

RAM



La variable `impresiones` es una variable de clase y es común a todas las instancias (una sola copia). Por eso imprime el valor 3.

Ejemplos de uso

```
public static void Main(string[] args)
{
    Auto a1 = new Auto("Ford", 1987);
    Auto a2 = new Auto("Peugeot", 1995);
    Auto a3 = new Auto("Fiat", 2008);
    a1.imprimir();
    a2.imprimir();
    a3.imprimir();
    Console.WriteLine(Auto.impresiones);
    a1.impresiones++;
    Auto.marca = "Renault";
    Auto.imprimir();
}
```

Esta sentencia produce error. Los miembros de clase no pueden ser referenciados con instancias

Puesta en común

Modifique la clase Hora agregándole el huso horario y escribe un programa de aplicación (Main) que permita imprimir por consola:

**xx HORAS, yy MINUTOS Y zz SEGUNDOS – (GTM-3),
donde GTM-3 corresponde al huso horario de Buenos Aires -
Argentina**

- Qué tipo de campo o miembro será huso horario?
- Defina los métodos necesarios para consultar y modificar dicho campo. Qué tipos de métodos son?

Programación orientada a objetos

¿Recuerdan los ArrayList?

`ArrayList a = new ArrayList();`

`double d = 4.67;`

`a.Add(d);`

`int[] i = new int[] {1,2,3,4,5,6,7};`

`a.AddRange(i);`

`a.Insert(4, "Hola");`

`Console.WriteLine(a.Count);`

ArrayList es una clase y por eso hay que instanciar objetos ArrayList antes de poder usarlos

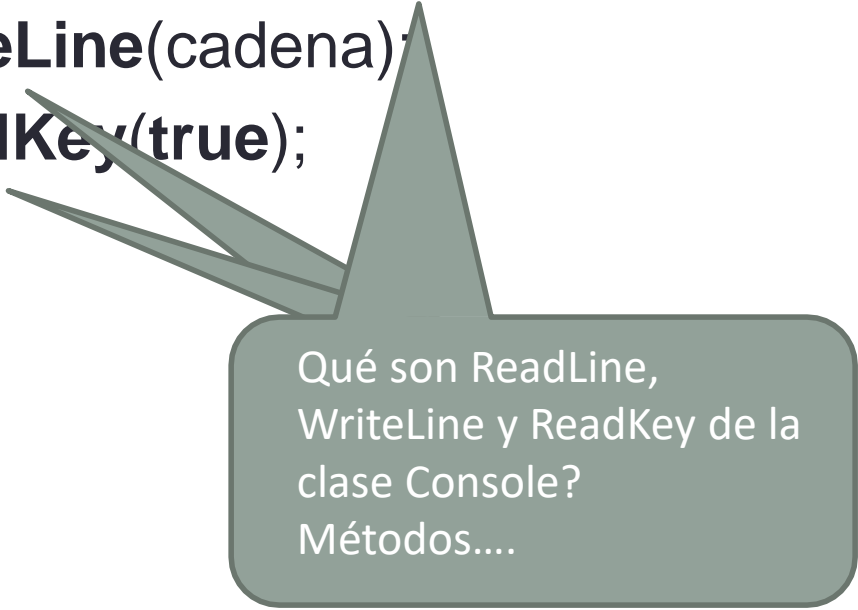
Add, AddRange e Insert son métodos de la clase ArrayList que definen el comportamiento de los objetos de esta clase

Count es una variable de instancia pública de ArrayList

Programación orientada a objetos

¿Recuerdan la clase Console?

```
string cadena = Console.ReadLine();  
Console.WriteLine(cadena);  
Console.ReadKey(true);
```



Qué son ReadLine,
WriteLine y ReadKey de la
clase Console?
Métodos....

Puesta en común

Defina una clase llamada "Mascota" con cuatro campos: nombre de la mascota, especie (perro, gato, tortuga, etc.), edad en años y nombre del dueño.

Defina un constructor que reciba solo la especie.

Defina un constructor que reciba el nombre de la mascota y la especie.

Defina un constructor que reciba los cuatro campos.

¿Cómo se llama esta propiedad que permite definir varios constructores con el mismo nombre?

```
class Mascota{  
    private string nombre, nombreDelDueño, especie;  
    private int edad;  
  
    public Mascota(string esp){  
        especie = esp;  
    }  
  
    public Mascota(string nom, string esp){  
        nombre = nom;  
        especie = esp;  
    }  
  
    public Mascota(string nom, string esp, string dueño, int e){  
        nombre = nom;  
        nombreDelDueño = dueño;  
        especie = esp;  
        edad = e;  
    }  
}
```

Agregue a la clase "Mascota" un **método de instancia** llamado "hablarConElDueño" que dependiendo de la especie haga su sonido característico (ladrar, maullar, relinchar, etc,)

```
class Mascota{  
    ...  
  
    public void hablarConElDueño(){  
  
        switch(especie){  
            case "perro": Console.WriteLine("Estoy ladrando"); break;  
            case "gato": Console.WriteLine("Estoy maullando"); break;  
            case "caballo": Console.WriteLine("Estoy relinchando"); break;  
            case "canario": Console.WriteLine("Estoy piando"); break;  
            case "conejo": Console.WriteLine("Estoy chillando"); break;  
            default:  
                Console.WriteLine("Soy un animal que no habla");  
                break;  
        }  
    }  
}
```


Haga un programa que instancie diferentes mascotas, las guarde en un ArrayList y luego recorra la colección haciendo "hablar" a las mascotas

```
ArrayList mascotas = new ArrayList();  
Mascota m;
```

```
m = new Mascota ("Fufu", "conejo");  
mascotas.Add(m);
```

```
m = new Mascota ("Firulai", "gato", "Pedro", 5);  
mascotas.Add(m);
```

```
mascotas.Add(new Mascota ("Fido", "perro"));  
mascotas.Add(new Mascota ("Manuelita", "tortuga", "Elena", 4));  
mascotas.Add(new Mascota ("Dory", "pez"));  
mascotas.Add(new Mascota ("Silver", "caballo", "Llanero", 10));  
mascotas.Add(new Mascota ("Hedwig", "lechuza", "Harry", 5));
```

```
string nom = "Tweety", especie = "canario";  
m = new Mascota (nom, especie);  
mascotas.Add(m);
```

```
foreach(Mascota mm in mascotas)  
    mm.hablarConElDueño();
```

```
ArrayList mascotas = new ArrayList();  
Mascota m;  
m = new Mascota ("Fufu", "conejo");  
mascotas.Add(m);  
m = new Mascota ("Firulai", "gato", "F");  
mascotas.Add(m);
```

A cada instancia de la clase Mascota le invocamos el método hablarConElDueño(). Cada instancia responderá distinto según la especie.

```
mascotas.Add(new Mascota ("Dory", "pez");  
mascotas.Add(new Mascota ("Silver", "ca", "Llanero", 10));  
mascotas.Add(new Mascota ("Hedwig", "serpiente", "Harry", 5));  
string nom = "Tweety", especie = "can";  
m = new Mascota (nom, especie);  
mascotas.Add(m);
```

```
foreach(Mascota mm in mascotas)  
    mm.hablarConElDueño();
```

Por qué usamos un foreach?

```
ArrayList mascotas = new ArrayList();
Mascota m;
m = new Mascota ("Fufu", "conejo");
mascotas.Add(m);
m = new Mascota ("Firulai", "gato", "Pedro", 5);
mascotas.Add(m);
mascotas.Add(new Mascota ("Fido", "perro"));
mascotas.Add(new Mascota ("Manuelita", "tortuga", "Elena", 4));
mascotas.Add(new Mascota ("Dory", "pez"));
mascotas.Add(new Mascota ("Silver", "caballo", "Llanero", 10));
mascotas.Add(new Mascota ("Hedwig", "lechuza", "Harry", 5));
string nom = "Tweety", especie = "canario";
m = new Mascota (nom, especie);
mascotas.Add(m);

foreach(Mascota mm in mascotas)
    mm.hablarConElDueño();
```

```
Estoy chillando
Estoy maullando
Estoy ladrando
Soy un animal que no habla
Soy un animal que no habla
Estoy relinchando
Soy un animal que no habla
Estoy piando
```

Muchas gracias

