

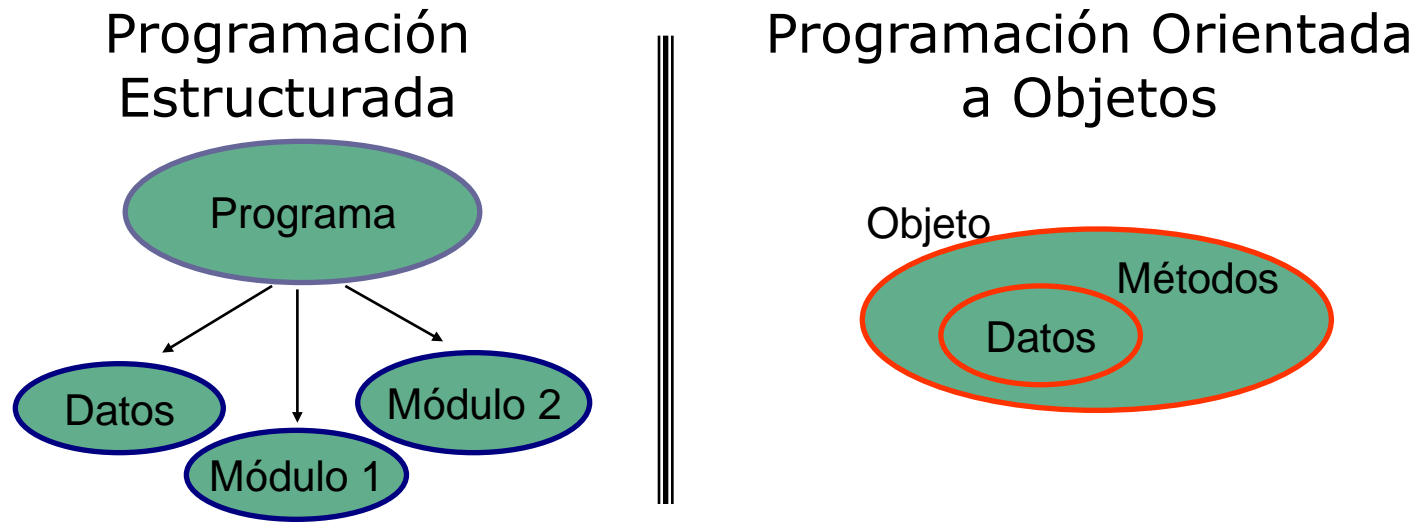
C#

Classes

¿Qué es la Programación Orientada a Objetos?

- Es una manera de construir Software. Es un **paradigma de programación**.
- 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.

¿Qué es la Programación Orientada a Objetos?



Prestar atención a los **verbos** de las especificaciones del sistema a construir si persigue un código **procedimental**, o los **sustantivos** si el **objetivo** es un programa **orientado a objetos**".

Ventaja de la Programación Orientada a Objetos

- **En el análisis y diseño:** Se puede especificar el problema usando un **vocabulario familiar a los usuarios** sin preparación técnica. El software se construye usando objetos que pertenecen a clases con las que el usuario está familiarizado
- **En la implementación:** Favorece la **reusabilidad** gracias a la **herencia** y la **modularización** y el **mantenimiento** gracias al encapsulamiento (ocultamiento de la información)

¿Qué es una clase?

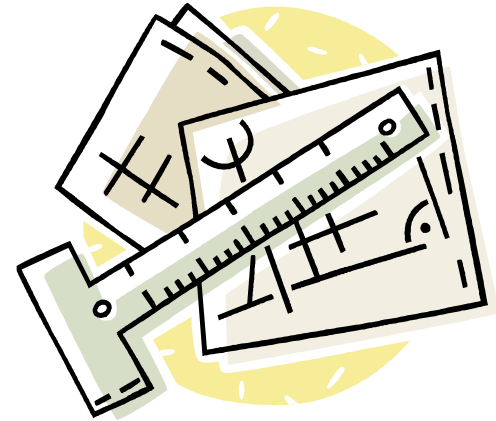
- Las clases son declaraciones de objetos. Esto quiere decir que la definición de un objeto es la Clase.
- Clasificación en base a comportamiento y atributos comunes

¿Qué es una clase?

- Una clase es una construcción estática que **describe un comportamiento común y atributos** (que toman distintos estados).
- Su formalización es a través de una estructura de datos que **incluye datos y funciones**, llamadas métodos. Los métodos son los que definen el comportamiento.

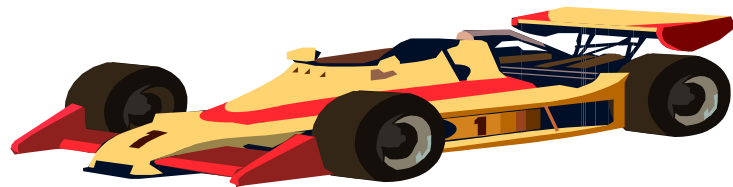
¿Qué es una clase?

- Construcción que Describe:
 - Comportamiento común
 - Atributos [estado]
- Incluye:
 - Funciones o métodos
 - Datos



Clases

Qué es lo que tienen en común?



Modelo

Marca

Color

Velocidad

Acelerar

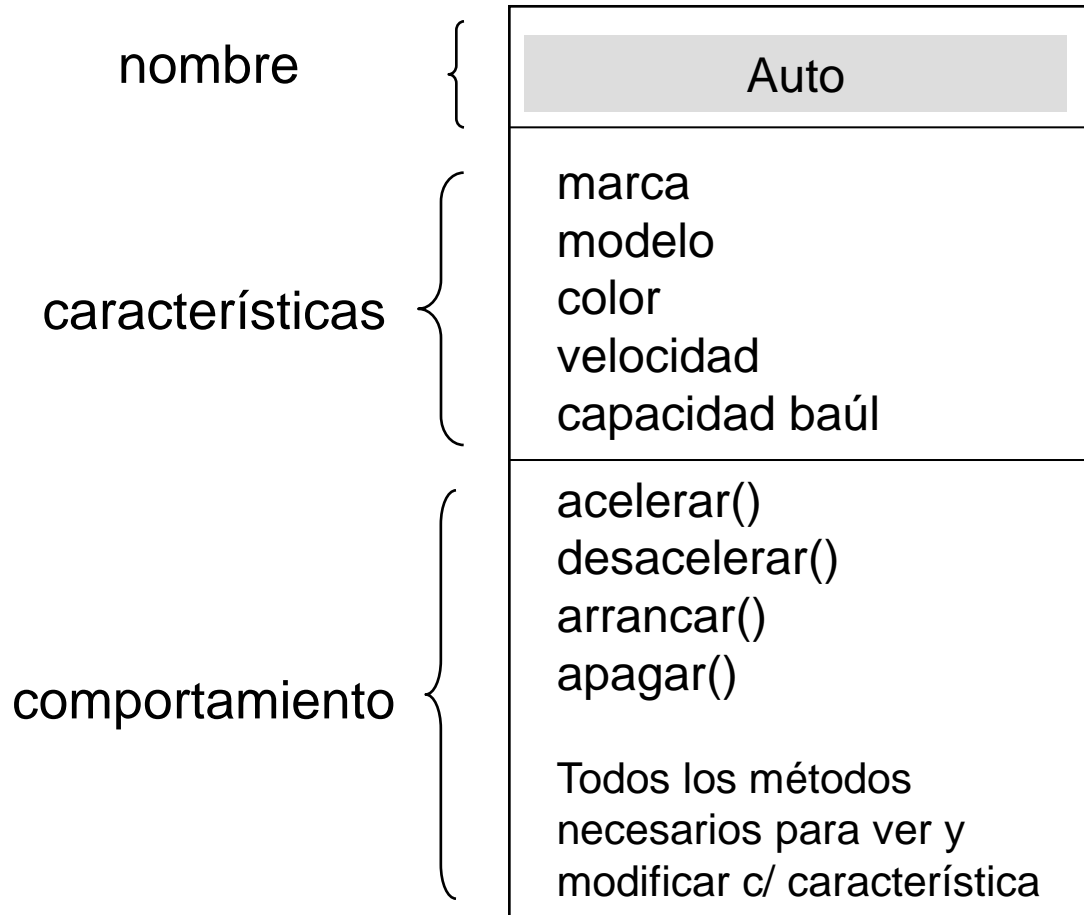
Desacelerar

Apagar

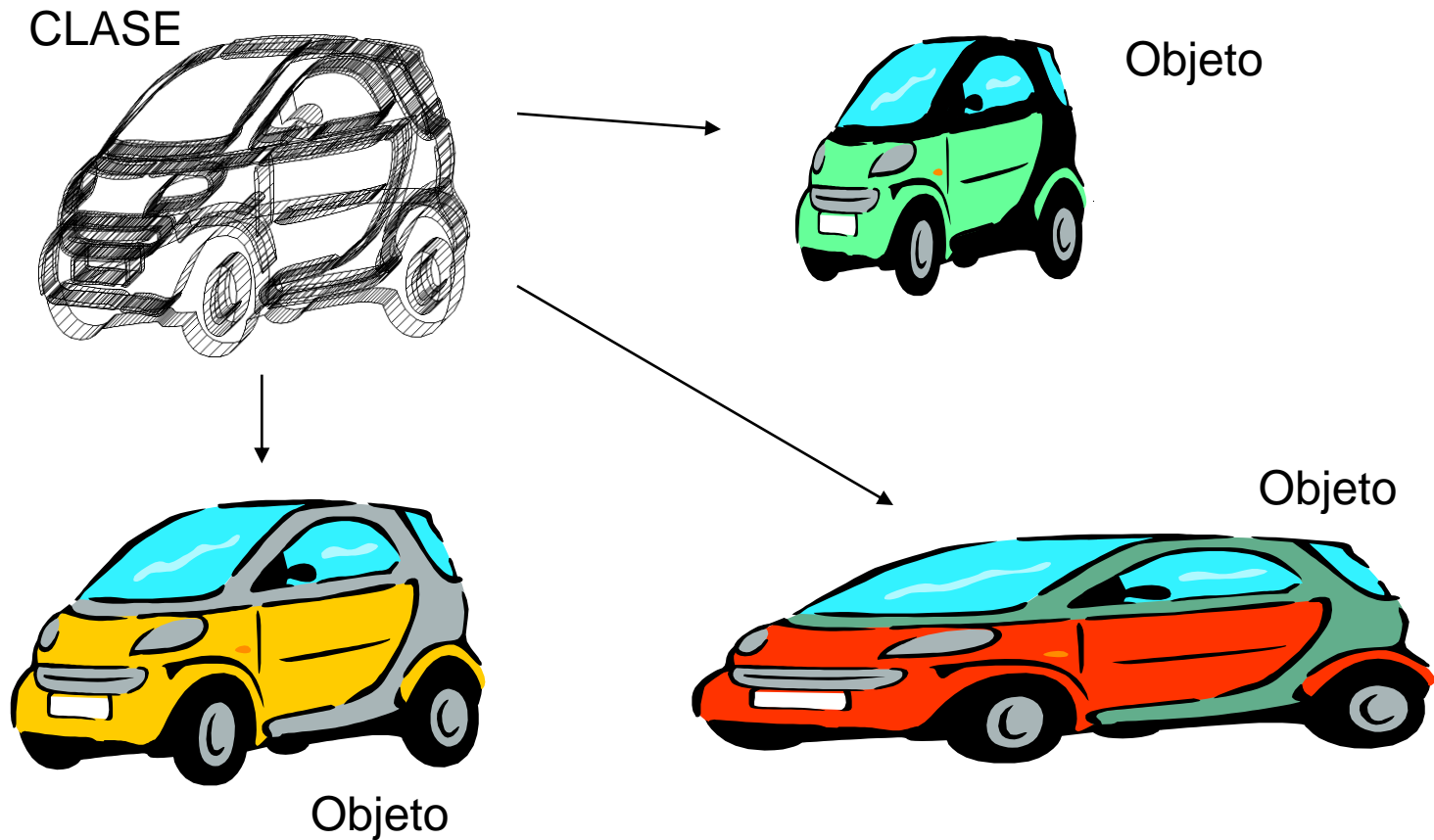
Arrancar

Se podría encontrar una forma de definir “algo” que encapsule las características y comportamiento comunes

Clases



¿Qué es un objeto?



¿Qué es un objeto?

- Instancia de una clase
- Cuando se crea una instancia (generalmente se utiliza el operador **new**) se debe especificar la clase a partir la cual se creará por ejemplo: **new StringBuilder()** .
- Por ejemplo, un objeto de la clase fracción es por ejemplo $\frac{2}{7}$. El concepto o definición de fracción sería la clase, pero cuando ya estamos hablando de una fracción en concreto $\frac{3}{5}$, $\frac{8}{10}$ o cualquier otra, la llamamos objeto.

Clases en C#

Sintaxis de definición de clases

La sintaxis básica para definir una clase es la que a continuación se muestra:

```
class <nombreClase>
{
    <miembros>
}
```

Los **miembros** de una clase son los datos y métodos de los que van a disponer todos los objetos de la misma

Clases en C#

Ejemplo de definición de una clases:

```
class Auto{  
}
```

Clases en C#

```
using System;  
class Programa{  
    static void Main(){  
    }  
}
```

Crear una aplicación de consola con el siguiente código fuente

```
class Auto{  
}
```

P. Cuántas clases hay?

R. Hay dos clases: Programa y Auto

P. Cuántos miembros tienen cada clase?

R. Hay un único miembro en la clase Programa, el método Main

Clases en C#

```
using System;  
class Programa{  
    static void Main(){  
        Auto a; // se declara una variable de tipo Auto  
    }  
}  
  
class Auto{  
}
```

P. Cuántos objetos ha creado (instanciado) el método Main de Programa ?

R. Ninguno, aún no se ha instanciado ningún objeto, sólo se ha declarado una variable de tipo Auto

Clases en C#

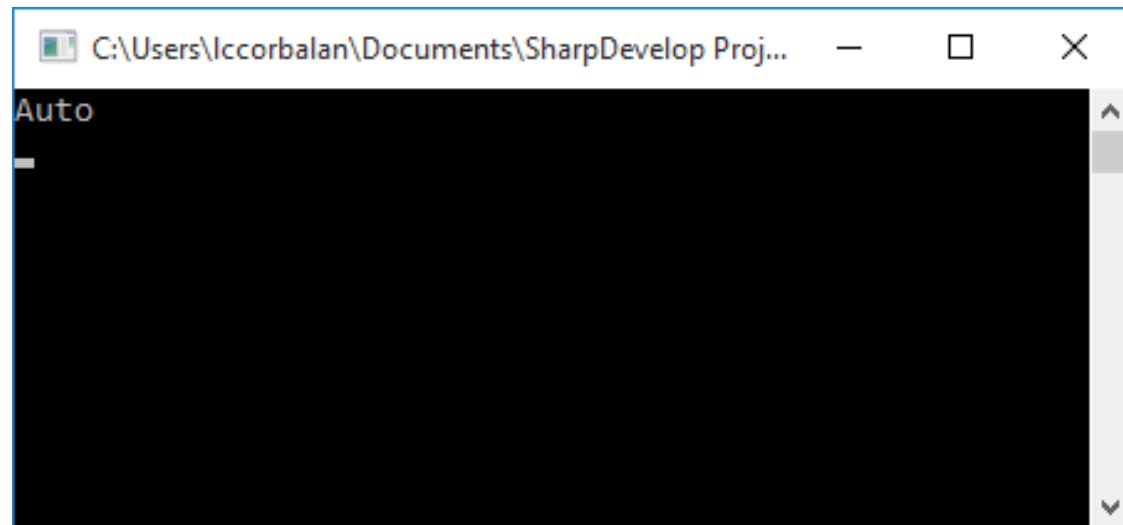
```
using System;
class Programa{
    static void Main(){
        Auto a; // se declara una variable de tipo Auto
        a = new Auto(); // se crea un objeto Auto (instanciación)
    }
}

class Auto{
}
```


Clases en C#

```
using System;
class Programa{
    static void Main(){
        Auto a; // se declara una variable de tipo Auto
        a = new Auto(); // se crea un objeto Auto (instanciación)
        Console.WriteLine(a); // se imprime el objeto
        Console.ReadKey(true);
    }
}

class Auto{
}
```



Clases en C# - Campos

- **Campos**: (variables de instancia) es un dato común a todos los objetos de una determinada clase.
- Se definen dentro de la clase con la siguiente sintaxis:

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

Clases en C# - Campos

Ejemplo definición de clase `Auto` con dos campos (`Marca` y `Modelo`)

```
class Auto{  
    string Marca;  
    int Modelo;  
}
```

Clases en C# - Campos

Para acceder a un campo de un determinado objeto se usa la sintaxis:


```
<objeto>.<campo>
```

Por ejemplo, si el campo es accesible, la siguiente instrucción cambia el valor del `Modelo` de un objeto `Auto` instanciado en una variable `a`:

```
a.Modelo = 2001;
```

Clases en C# - Campos

```
class Programa{  
    static void Main(){  
        Auto a; // se declara una variable de tipo Auto  
        a = new Auto(); // se crea un objeto Auto (instanciación)  
        a.Marca = "Fiat";  
        a.Modelo = 2000;  
        Console.WriteLine(a);  
        Console.ReadKey(true);  
    }  
}
```



Marca y Modelo no son accesibles debido a su nivel de protección.

Los miembros de una clase son privados por defecto.

Clases en C# - Campos

Hacemos públicos los dos campos de la clase `Auto` (`Marca` y `Modelo`)

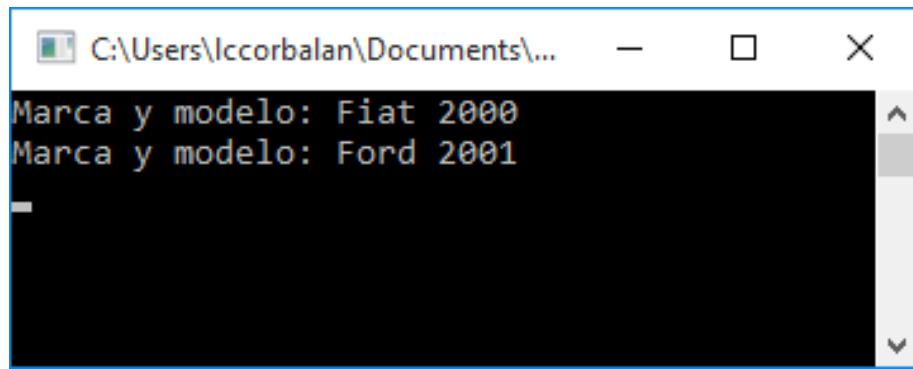
```
class Auto{  
    public string Marca;  
    public int Modelo;  
}
```

Clases en C# - Campos

```
class Programa{  
    static void Main(){  
        Auto a; // se declara una variable de tipo Auto  
        a = new Auto(); // se crea un objeto Auto (instanciación)  
        a.Marca = "Fiat";  
        a.Modelo = 2000;  
        Console.WriteLine("Marca y modelo: {0} {1}",a.Marca,a.Modelo);  
        Console.ReadKey(true);  
    }  
}
```

Clases en C# - Campos

```
class Programa{
    static void Main(){
        Auto a; // se declara una variable de tipo Auto
        a = new Auto(); // se crea un objeto Auto (instanciación)
        a.Marca = "Fiat";
        a.Modelo = 2000;
        Console.WriteLine("Marca y modelo: {0} {1}",a.Marca,a.Modelo);
        Auto b = new Auto();
        b.Modelo = 2001;
        b.Marca = "Ford";
        Console.WriteLine("Marca y modelo: {0} {1}",b.Marca,b.Modelo);
        Console.ReadKey(true);
    }
}
```



```
C:\Users\lccorbalan\Documents\...
Marca y modelo: Fiat 2000
Marca y modelo: Ford 2001
```


Clases en C# - Métodos

Son el equivalente a las **funciones** y **procedimientos** de Pascal pero asociados a una clase de objetos determinada.

Dentro de los métodos puede accederse a todos los campos (incluido los privados) de la clase.

Los métodos permiten manipular los datos almacenados en los objetos.

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

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

Clases en C# - Métodos

Ejemplo: Definiendo el método `imprimir()` en la clase `Auto`

```
class Auto{  
    public string Marca;  
    public int Modelo;  
    public void Imprimir(){  
        Console.WriteLine("Marca y modelo: {0} {1}",Marca,Modelo);  
    }  
}
```

Clases en C# - Métodos

Ahora se puede reemplazar la instrucción:

```
Console.WriteLine("Marca y modelo: {0} {1}", a.Marca, a.Modelo);
```

Por la instrucción:

```
a.Imprimir();
```

Clases en C# - Métodos

```
class Programa{  
    static void Main(){  
        Auto a; // se declara una variable de tipo Auto  
        a = new Auto(); // la instanciamos  
        a.Marca = "Fiat";  
        a.Modelo = 2000;  
        a.Imprimir();  
        Auto b = new Auto();  
        b.Modelo = 2001;  
        b.Marca = "Ford";  
        b.Imprimir();  
        Console.ReadKey(true);  
    }  
}
```

Le acabamos de dar la responsabilidad de imprimirse a los propios autos. Es una mejor asignación de competencias, pues deberíamos evitar el acceso a la representación interna de los objetos desde afuera de los mismos

Clases en C# - Sobrecarga de Métodos

Una clase puede tener más de un método con el mismo nombre siempre que sus **firmas** sean diferentes

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

El tipo de retorno no es parte de la **firma**

Los nombres de los parámetros tampoco son parte de la **firma**

Clases en C# - Sobrecarga de Métodos

Ejemplos de sobrecarga válidos

```
void procesar(int valor)
```

```
void procesar(float valor)
```

```
void procesar(double valor)
```

```
void procesar(int valor1, double valor2)
```

```
void procesar(double valor1, int valor2)
```

```
void procesar(out double valor)
```

Clases en C# - Sobrecarga de Métodos

Ejemplos de sobrecarga inválidos

```
void procesar(int valor)
```

```
int procesar(int valor)
```

El tipo de retorno no es parte de la firma

Clases en C# - Sobrecarga de Métodos

Ejemplos de sobrecarga inválidos

```
void procesar(int valor1)
```

```
void procesar(int valor2)
```

El nombre de los parámetros
no es parte de la firma

Clases en C# - Sobrecarga de Métodos

Ejemplos de sobrecarga inválidos

```
void procesar(ref int valor)
```

```
void procesar(out int valor)
```

La diferencia no puede ser únicamente **ref** y **out**

Clases en C# - Sobrecarga de Métodos

Agregar los siguientes miembros a la clase Auto

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

Por convención los miembros privados empiezan con minúscula

Sobrecargar
del método
Acelerar

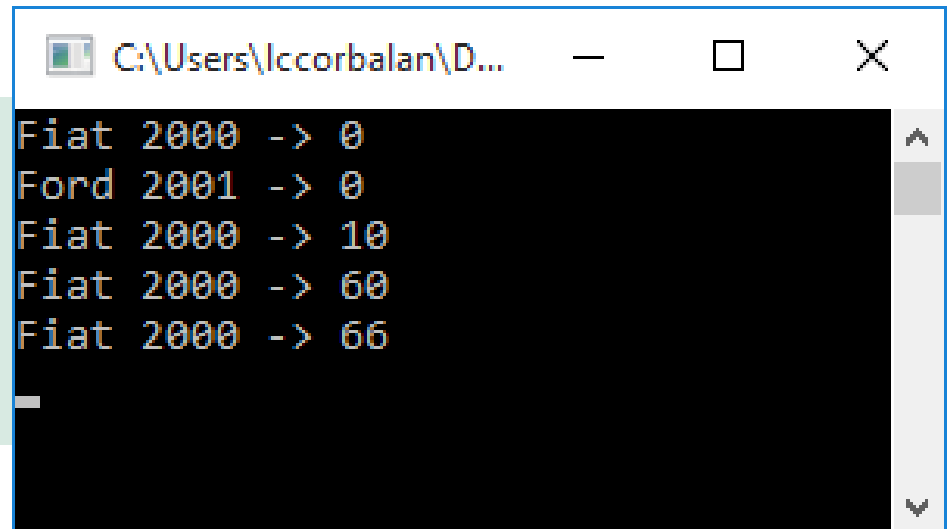
Clases en C# - Sobrecarga de Métodos

Modificar el método Imprimir() para que también se muestre la velocidad

```
public void Imprimir(){  
    Console.WriteLine("{0} {1} -> {2}",  
                        Marca, Modelo, velocidad);  
}
```

Clases en C# - Sobrecarga de Métodos

```
class Programa{
    static void Main(){
        Auto a; // se declara una variable de tipo Auto
        a = new Auto(); // se crea un objeto Auto (instanciación)
        a.Marca = "Fiat";
        a.Modelo = 2000;
        a.Imprimir();
        Auto b = new Auto();
        b.Modelo = 2001;
        b.Marca = "Ford";
        b.Imprimir();
        a.Acelerar();
        a.Imprimir();
        a.Acelerar(50);
        a.Imprimir();
        a.Acelerar(1.1);
        a.Imprimir();
        Console.ReadKey(true);
    }
}
```



```
C:\Users\lccorbalan\D...
Fiat 2000 -> 0
Ford 2001 -> 0
Fiat 2000 -> 10
Fiat 2000 -> 60
Fiat 2000 -> 66

```

Clases en C# - Constructores de instancia

Una estrategia muy utilizada para establecer el estado de un objeto (generalmente **asignando campos**) es hacerlo en el momento de su **creación** a través del **pasaje de parámetros**.

Un **constructor de instancia** definido en una clase es un **métodos especial** que contiene código a ejecutar en el **momento de la instanciación** de un objeto de esa clase.

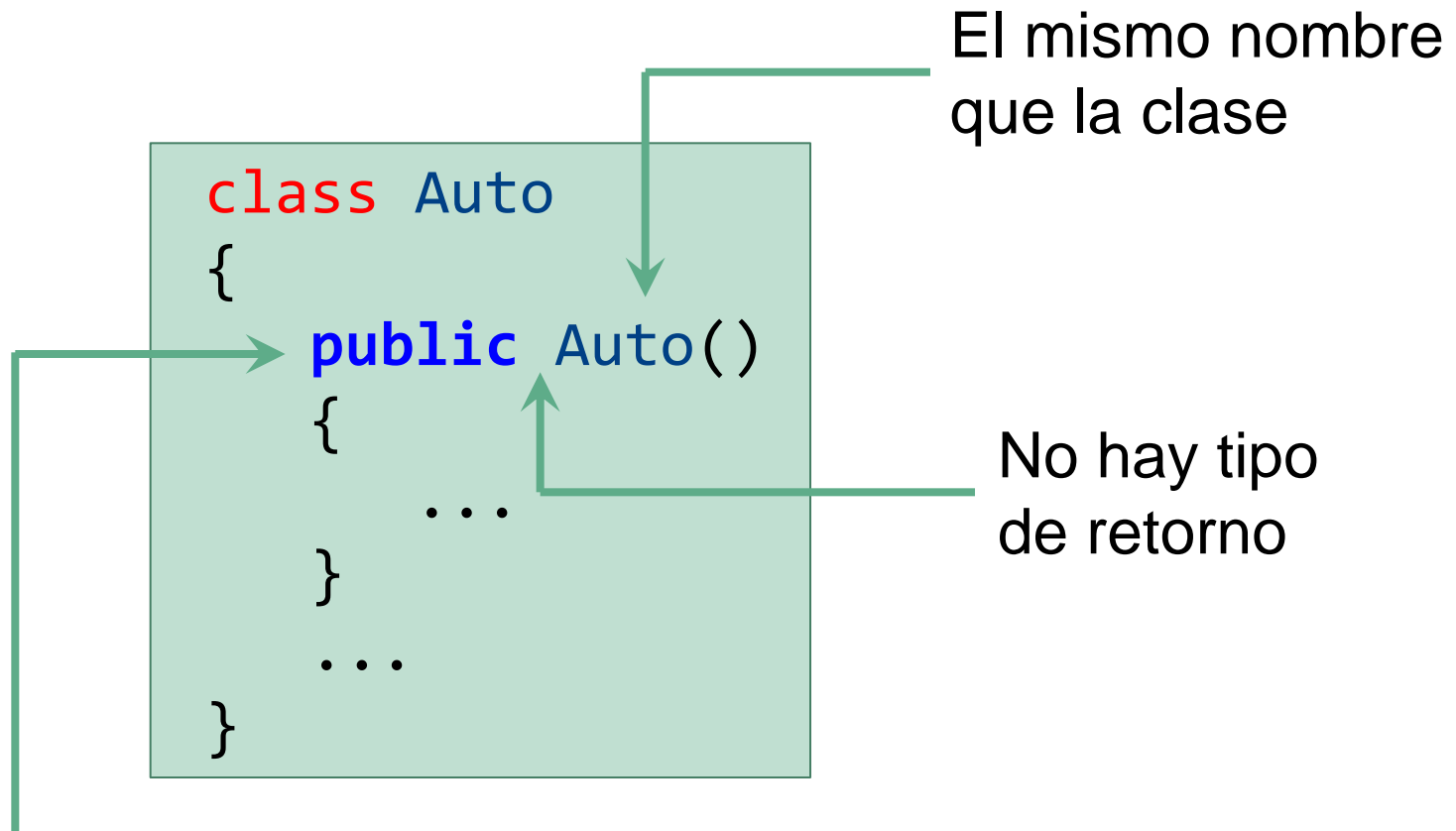
Clases en C# - Constructores de instancia

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. Es decir, se usa la sintaxis:

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

Si se desea poder crear instancias de la clase desde fuera de la misma es necesario declarar el constructor público

Clases en C# - Constructores de instancia



Declarado `public` para que pueda ser invocado desde fuera de la clase

Seguimos trabajando con la clase Auto

```
class Auto{
    public string Marca;
    public int Modelo;
    private double velocidad=0;
    public void Imprimir()
    {
        Console.WriteLine("{0} {1} -> {2}",
                           Marca, Modelo, velocidad);
    }

    ...
}
```


Seguimos trabajando con la clase Auto

```
class Auto{  
    private string marca;  
    private int modelo;  
    private double velocidad=0;  
    public void Imprimir()  
    {  
        Console.WriteLine("{0} {1} -> {2}",  
                             marca, modelo, velocidad);  
    }  
  
    ...  
}
```

Hacer privados los campos `marca` y `modelo`
(por convención, la primera letra de los miembros privados se escribe en minúscula)

Seguimos trabajando con la clase Auto

```
class Auto{  
    private string marca;  
    private int modelo;  
    private double velocidad=0;  
    public void Imprimir()  
    {  
        Console.WriteLine("{0} {1} -> {2}",  
                             marca, modelo, velocidad);  
    }  
}
```

```
public Auto(string mar, int mod)  
{  
    marca = mar;  
    modelo = mod;  
}
```

Agregar este constructor

...

Seguimos trabajando con la clase Auto

Ejecute y compruebe su funcionamiento

```
class Programa{  
    static void Main()  
    {  
        Auto a = new Auto("Fiat",2000);  
        a.Imprimir();  
        Console.ReadKey();  
    }  
}
```



Fiat 2000 -> 0

The screenshot shows a black console window with the text "Fiat 2000 -> 0" in a monospaced font. The text is colored: "Fiat" is blue, "2000" is orange, and "-> 0" is white. A vertical scrollbar is visible on the right side of the window.

Seguimos trabajando con la clase Auto

Tanto en métodos como en constructores es común utilizar **parámetros con el mismo nombre de aquellos campos que se desean asignar**. En este caso se utiliza **this** para diferenciar el campo del parámetro

```
public Auto(string marca, int modelo)
{
    this.marca = marca;
    this.modelo = modelo;
}
```

La palabra clave **this** usada en una clase es una referencia a la instancia corriente.

Seguimos trabajando con la clase Auto

Modifique la clase `Programa` para instanciar un objeto `Auto` sin parámetros, como veníamos haciendo antes de agregar el constructor

```
class Programa{  
    static void Main()  
    {  
        Auto a = new Auto("Fiat", 2000);  
        a.Imprimir();  
        Auto b = new Auto();  
        b.Imprimir();  
        Console.ReadKey();  
    }  
}
```



Agregar estas
líneas



Clases en C# - Constructores de instancia

Constructor por defecto

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

```
<nombreTipo>()  
{  
}
```

Si definimos un constructor, el compilador no incluye ningún otro constructor. Por ello, en el código anterior, la instrucción

```
Auto b = new Auto();
```

produce un error de compilación pues el constructor por defecto no existe más.

Clases en C# - Constructores de instancia



Constructor por defecto

¿Porqué puede ser útil esta estrategia del compilador de no incluir el constructor por defecto si nosotros agregamos uno explícitamente?

Porque eventualmente podemos querer forzar al usuario a utilizar determinado constructor garantizando así la consistencia de los objetos creados.

Clases en C# - Constructores de instancia

Sobrecarga de constructores

Al igual que los métodos, los constructores pueden ser sobrecargados.

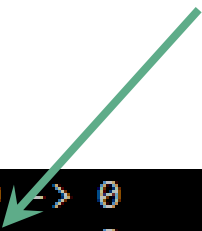
Para ello, es necesario que las firmas de todos los constructores sean diferentes entre sí.

Seguimos trabajando con la clase Auto

Ejercicio: Definir otro constructor para la clase `Auto` que permita utilizar a dicha clase de la siguiente manera:

```
class Programa{  
    static void Main()  
    {  
        Auto a = new Auto("Fiat",2000);  
        a.Imprimir();  
        Auto b = new Auto();  
        b.Imprimir();  
        Console.ReadKey();  
    }  
}
```

Año corriente



```
Fiat 2000 -> 0  
Ford 2016 -> 0
```

Seguimos trabajando con la clase Auto

SOLUCIÓN

```
public Auto()  
{  
    marca = "Ford";  
    modelo = DateTime.Today.Year;  
}
```

Cada vez que se instancie un `Auto` sin parámetros, se creará un auto marca "Ford" y modelo igual al año corriente.

Clases en C# - Constructores de instancia

Sobrecarga de constructores

En el encabezado de un constructor se puede invocar a otro constructor empleando **:this**.

Este constructor será invocado antes de que se ejecuten las instrucciones del cuerpo del constructor que lo invoca.

Seguimos trabajando con la clase Auto

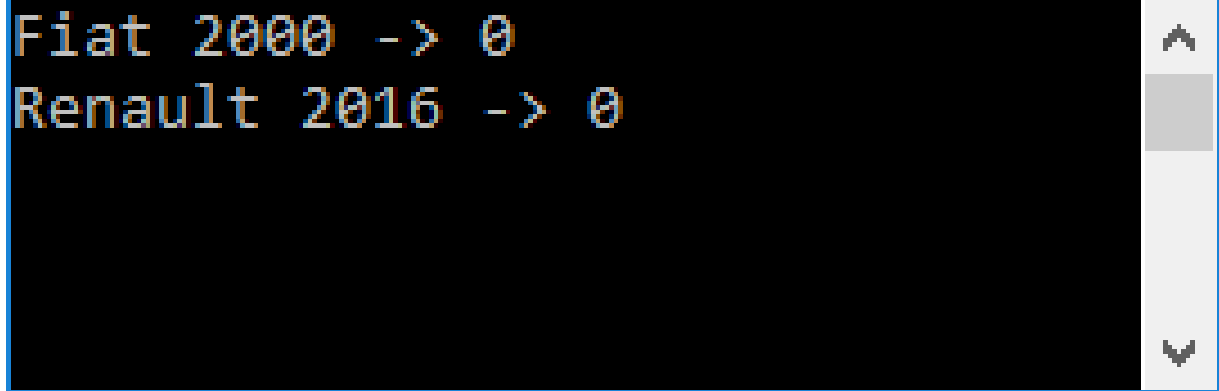
Agregamos un constructor a la clase `Auto` que permite indicar la marca del mismo. El modelo del `Auto` creado deberá ser igual al año corriente.

```
public Auto()  
{  
    marca = "Ford";  
    modelo = DateTime.Today.Year;  
}  
public Auto(string marca): this()  
{  
    this.marca = marca;  
}
```

Seguimos trabajando con la clase Auto

Ejecute y compruebe su funcionamiento

```
class Programa{  
    static void Main()  
    {  
        Auto a = new Auto("Fiat",2000);  
        a.Imprimir();  
        Auto b = new Auto("Renault");  
        b.Imprimir();  
        Console.ReadKey();  
    }  
}
```




```
Fiat 2000 -> 0  
Renault 2016 -> 0
```

Seguimos trabajando con la clase Auto

También pudo haberse definido de la siguiente forma:

```
public Auto(string marca, int modelo)
{
    this.marca = marca;
    this.modelo = modelo;
}
```



```
public Auto(string marca): this(marca,
                                DateTime.Today.Year)
{
}
```

En este caso el cuerpo del constructor está vacío, todo el trabajo se hace al invocar al constructor anterior

Repaso práctica 3

Excepciones

¿Indique la o las líneas que provocan una excepción

```
int x=0;  
Console.WriteLine(1.0/x);  
→ Console.WriteLine(1/x);
```

Repaso práctica 3

Excepciones

Suponiendo que metodo2 es invocado desde un método llamado metodo1.

¿ metodo1 se entera de la ocurrencia de la excepción ?

```
static void metodo2() {  
    byte b=255;  
    try{  
        b++;  
    }finally{  
        Console.WriteLine("bloque finally");  
    }  
}
```

Sí. Debido a que `try` no posee cláusula `catch`, la excepción se propaga al método que invocó `metodo2()`

Seminario .NET

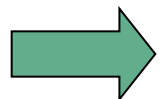
Autoevaluación 1

Indique cuál de todas las afirmaciones es verdadera respecto del siguiente fragmento de código:

```
Console.WriteLine(5/2);  
Console.WriteLine(5.0/2);  
Console.WriteLine(5/2.0);
```

1) Se imprime en la consola: 5/25.0/25/2.0

2) Se produce un error en tiempo de ejecución.



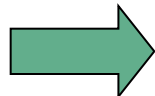
3) Se imprime en la consola: 22,52,5 (siendo la coma (,) el símbolo decimal elegido en la Configuración regional del Windows)

Indique cuál de todas las afirmaciones es verdadera respecto del siguiente programa (preste atención a los índices)

```
using System;
class HolaMundo
{
    public static void Main(String[] args)
    {
        Console.WriteLine(";Hola {0}!", args[1]);
        Console.ReadKey();
    }
}
```

1) Para evitar un posible error en tiempo de ejecución debe agregarse la condición: `if (args.Length > 0)` como primera línea del método `Main`

2) Hay al menos un error sintáctico que impide su compilación

 3) Una vez generado, debe invocarse con al menos 2 argumentos pasados como parámetro por la línea de comandos para evitar un error en tiempo de ejecución.

Indique cuál de todas las afirmaciones es verdadera respecto del siguiente programa

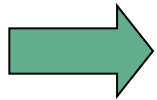
```
class Programa{
    static void Main() {
        int r=0;
        suma(10,20,r);
        System.Console.WriteLine(r);
    }
    static void suma(int n1,int n2,ref int result){
        result = n1+n2;
    }
}
```

1) Hay error en tiempo de ejecución

 2) Hay error de compilación en la invocación al método `suma`

3) Produce la siguiente salida por la consola: 30

Indique cual de las siguientes afirmaciones es verdadera:



1) C# es sólo uno de los varios lenguajes de programación existentes para la plataforma .Net

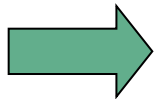
2) .Net es un lenguaje moderno de programación de alto nivel

3) El Common Language Runtime (CLR) es sólo uno de los varios lenguajes de programación existentes para la plataforma .Net

¿Qué salida produce por el consola el siguiente fragmento de código?

```
double d= 159.0/100;
```

```
Console.WriteLine("d={0:0.0}", d);
```



1) **d=1 , 6**

2) **d=1 , 5**

3) **d=1 , 59**