

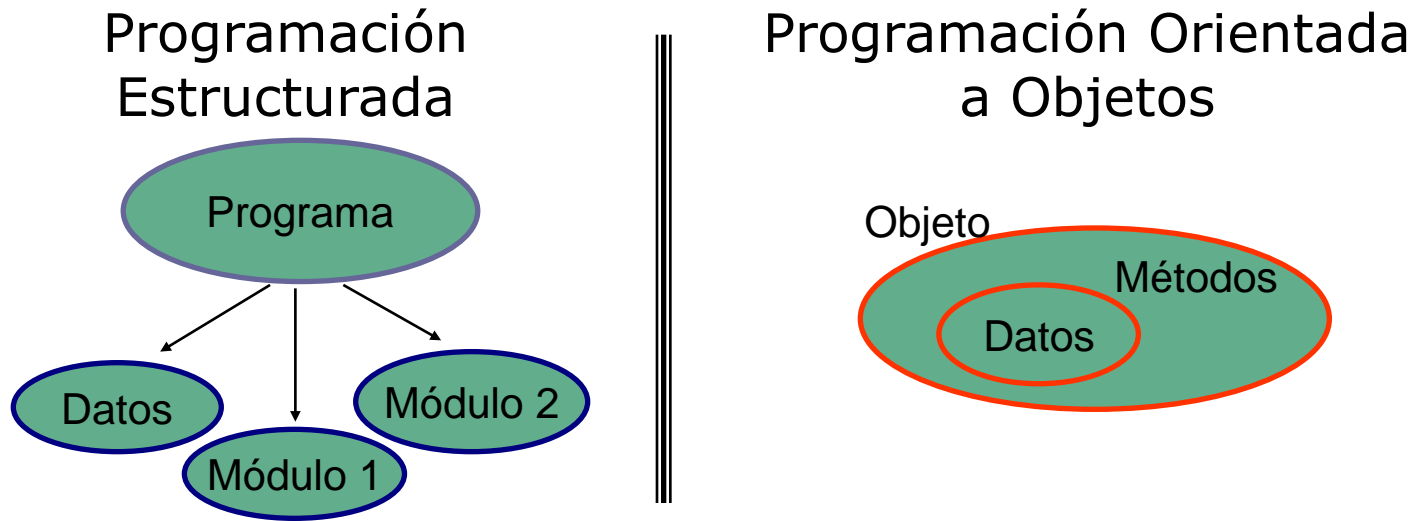
C#

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?

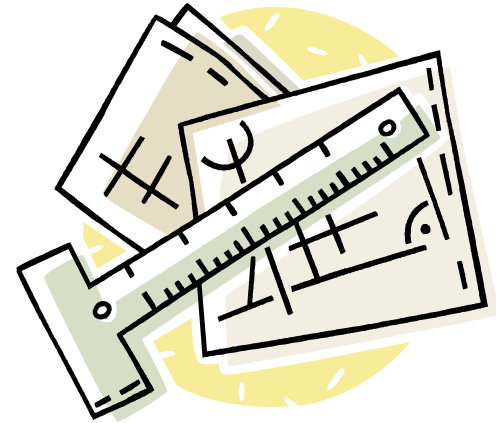
- Las clases son declaraciones de objetos. Esto quiere decir que la definición de un objeto es la Clase.
- Clasificación en base a comportamiento y atributos comunes

¿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.

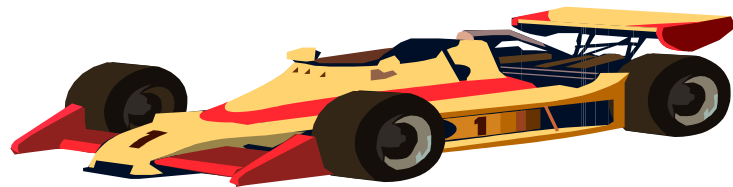
¿Qué es una clase?

- 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

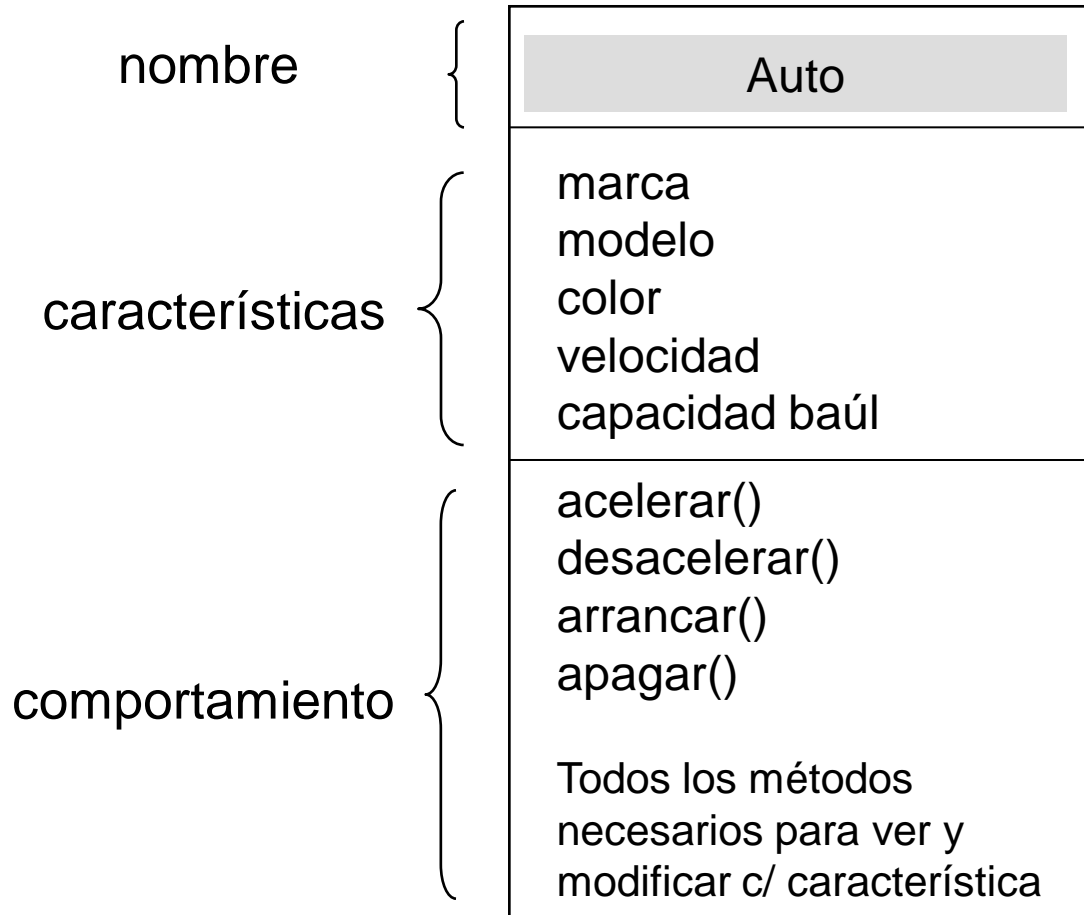
Desacelerar

Apagar

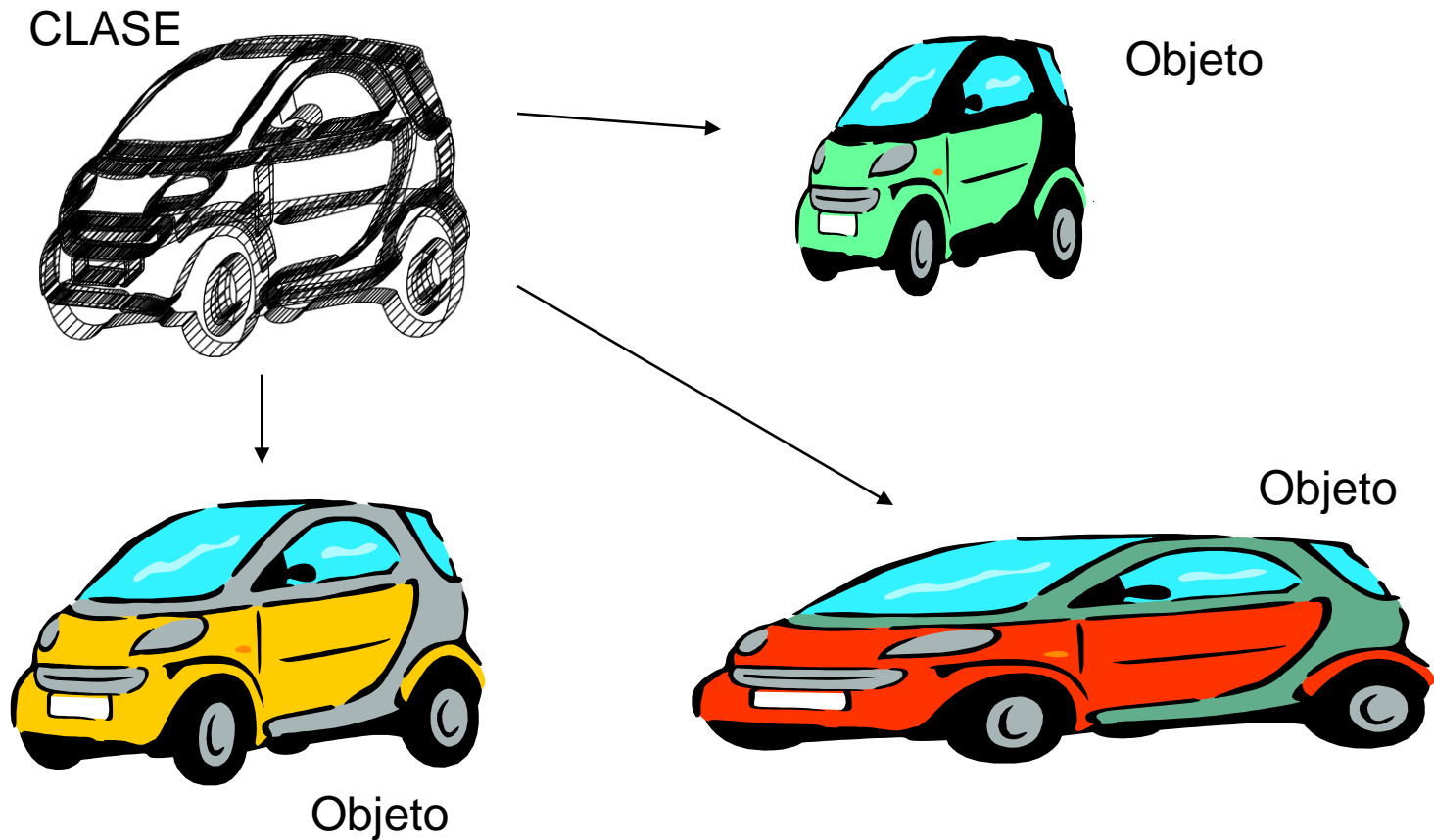
Arrancar

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(){
        Auto a; // declaramos una variable de tipo Auto
        a = new Auto(); // la instanciamos
        Console.WriteLine("Presione una tecla para continuar...");
        Console.ReadKey(true);
    }
}
```

```
class Auto{
}
```



The screenshot shows a Windows command prompt window with a dark background. The title bar at the top reads "E:\discoE\misDocumentos\SharpDevelop Projects\cla". The command prompt itself displays the text "Presione una tecla para continuar..." in a monospaced font. A white cursor is visible on the line below the text.

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


Clases en C# - Campos

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

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

Por ejemplo, para acceder al campo `modelo` de un objeto `Auto` llamado `a` y cambiar su valor por `2001` se haría:

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

Clases en C# - Campos

```
class Programa{  
    static void Main(){  
        Auto a;  
        a = new Auto();  
        a.Marca = "Fiat";  
        a.Modelo = 2000;  
        Auto b = new Auto();  
        b.Modelo = 2001;  
        b.Marca = "Ford";  
        Console.WriteLine("Marca y modelo: {0} {1}",a.Marca,a.Modelo);  
        Console.WriteLine("Marca y modelo: {0} {1}",b.Marca,b.Modelo);  
        Console.WriteLine("Presione una tecla para continuar...");  
        Console.ReadKey(true);  
    }  
}
```



The screenshot shows a Windows command prompt window with the following text:

```
c:\ E:\discoE\misDocumentos\SharpDevelop Projects\w  
Marca y modelo: Fiat 2000  
Marca y modelo: Ford 2001  
Presione una tecla para continuar...  
_
```

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`, se evitaría tener que acceder a sus variables de instancia desde el código fuera de la clase.

```
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

```
using System;
class Programa{
    static void Main(){
        Auto a;
        a = new Auto();
        a.Marca = "Fiat";
        a.Modelo = 2000;
        Auto b = new Auto();
        b.Modelo = 2001;
        b.Marca = "Ford";
        a.Imprimir();
        b.Imprimir();
        Console.ReadKey(true);
    }
}
```

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

Agregar los siguientes miembros a la clase Auto

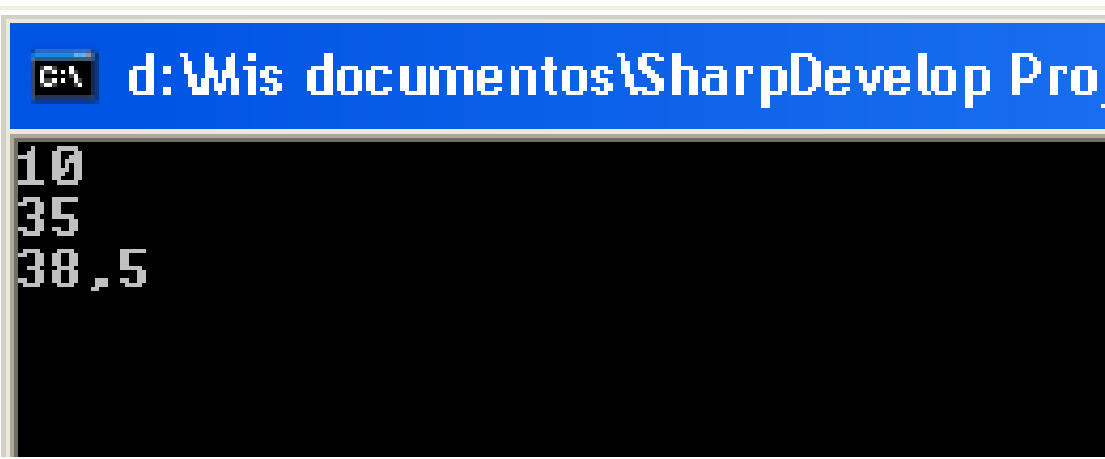
```
private double velocidad;  
public double Acelerar() {  
    return velocidad += 10;  
}  
public double Acelerar(int valor) {  
    return velocidad += valor;  
}  
public double Acelerar(double coeficiente) {  
    return velocidad *= coeficiente;  
}
```

Sobrecargar
del método
Acelerar



Clases en C# - Sobrecarga de Métodos

```
static void Main() {  
    Auto a = new Auto();  
    Console.WriteLine(a.Acelerar());  
    Console.WriteLine(a.Acelerar(25));  
    Console.WriteLine(a.Acelerar(1.1));  
    Console.ReadKey(true);  
}
```



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

```
C:\> d:\Wis documentos\SharpDevelop Pro  
10  
35  
38,5
```

Clases en C# - Constructores

Una estrategia muy utilizada para **asignar campos** de un objeto es hacerlo en el momento de su **creación** a través del **pasaje de parámetros**.

Un **constructor** definido en una clase es un método especial que contiene código a ejecutar cada vez que se crea un objeto de esa clase.

Clases en C# - Constructores

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>  
}
```

Clases en C# - Constructores

Redefinimos la clase `Auto` haciendo privadas las variables de instancia y definiendo un constructor adecuado para poder setearlas en el momento de crear el objeto.

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

Por convención utilizamos términos en minúscula para los miembros privados de una clase pero no es obligatorio.

La palabra clave **this** se utiliza para diferenciar las variables de instancia de los parámetros locales del mismo nombre

Clases en C# - Constructores



Ahora se reescribe el método `Main` de la siguiente forma:

```
static void Main() {  
    Auto a = new Auto("Fiat", 2000);  
    Auto b = new Auto("Ford", 2001);  
    a.Imprimir();  
    b.Imprimir();  
    Console.WriteLine("Presione una tecla para continuar");  
    Console.ReadKey(true);  
}
```

Clases en C# - Constructores

Ahora se reescribe el método `Main` de la siguiente forma:

```
static void Main() {  
    Auto a = new Auto("Fiat", 2000);  
    Auto b = new Auto("Ford", 2001);  
    Auto c = new Auto();  
    a.Imprimir();  
    b.Imprimir();  
    Console.WriteLine("Presione una tecla para continuar");  
    Console.ReadKey(true);  
}
```



Clases en C# - Constructores

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 `c=new Auto ()` ; da error de compilación pues el constructor por defecto no existe más.


Clases en C# - Constructores

Sobrecarga de constructores

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

Clases en C# - Constructores

```
class Auto{  
    private string marca;  
    private int modelo;  
    public void Imprimir(){  
        Console.WriteLine("Marca y modelo: {0} {1}",  
                           marca, modelo);  
    }  
    public Auto(string marca, int modelo){ //constructor  
        this.marca = marca;  
        this.modelo = modelo;  
    }  
    public Auto(string marca, string modelo){ //constructor  
        this.marca = marca;  
        this.modelo = Convert.ToInt32(modelo);  
    }  
}
```



Agregue este constructor

Clases en C# - Constructores

Sobrecarga de constructores

Ahora podemos hacer uso de ambos constructores codificando cosas como esta:

```
Auto a = new Auto("Fiat", "1999");  
Auto b = new Auto("Ford", 2000);
```

En el primer caso se llama al constructor 2 puesto que se pasan dos strings como parámetros. En el segundo caso se invoca el constructor 1

Repaso práctica 3

Excepciones

¿Qué línea provoca una excepción?

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

Repaso práctica 3

Excepciones

¿Dónde se captura la excepción para ser tratada?

```
static void metodo1() {  
    byte b=255;  
    try{  
        b++;  
    } finally{  
        Console.WriteLine("bloque finally");  
    }  
}
```

Debido a que `try` no posee cláusula `catch`, la excepción se propaga al método que invocó `metodo1()`