

Clases y objetos

Clases y objetos

¿Qué es una CLASE?

- Una clase es una estructura o plantilla que define las características y comportamiento que van a tener las instancias de esa clase.
- Cada clase usualmente se codifica en un archivo separado. (cs)
- Para describir la estructura se utilizan atributos
- Para describir el comportamiento utiliza métodos

¿Cómo se escribe una clase en C #?

```
public class Auto{  
  
}
```

La estructura y comportamiento se especifican entre las llaves de apertura y cierre de la clase

Clases y objetos

¿Qué es un objeto?

Un objeto es una instancia de una clase; una entidad tangible o visible de un problema.

¿Qué características tiene un objeto?

Tienen:

- Una identidad (son únicos aunque tengan los mismos datos)
- Estado, sus datos pueden ser modificados durante su ciclo de vida; está determinado por el valor de sus atributos
- Comportamiento: determinado por los métodos

Clases y objetos

¿Qué son los atributos?

Los atributos representan la información que nos interesa guardar de un objeto. Pueden ser de instancia o de clase.

Ejemplo:

Clase Auto que tiene los atributos:

- matricula(texto)
- marca(texto)
- modelo(texto)
- año de fabricación(número)

Clases y objetos

¿Cómo se declaran los atributos en C#?

visibilidad tipoDato nombreAtributo;

Ejemplo de declaración de atributos en C#:

```
private string _matricula;  
private string _marca;  
private string _modelo;  
private int _añoFabricacion;
```

Se escriben en minúscula.

Properties

Son métodos que permiten al usuario recuperar (get) o indicar un valor a un atributo del objeto (set)

Puede declararse de solo lectura es decir no se declara el set; esto es cuando no se desea asignar valor a un atributo.

Ejemplo

```
public string Marca{  
    get { return _marca; }  
    set { _marca= value; }  
}
```

Clase Auto

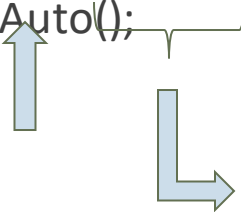
```
public class Auto{
    // Atributos
    private string _matricula;
    private string _marca;
    private string _modelo;
    private int _añoFabricacion;

    // Properties
    public string Matricula{
        get { return _matricula; }
        set { _matricula= value; }
    }
    public int AñoFabricacion{
        get { return _añoFabricacion; }
        set { _añoFabricacion= value; }
    }
    .....
}
```

Creando Objetos - new

- Para crear una instancia de la clase Auto (objeto), debemos utilizar el operador new de C#.
- El objeto se crea en memoria y para poder utilizarlo, podemos por ejemplo guardar la referencia a éste en una variable declarada con el tipo de la clase que vamos a instanciar.

```
Auto unAuto = new Auto();
```



Al crearse la instancia se se invoca inmediatamente al método Constructor

Clases y objetos

Método constructor

- Se utiliza para inicializar una instancia de esa clase (objeto) y dentro del mismo se pueden inicializar los atributos de ese objeto instanciado.
- Todas las clases tienen un constructor por defecto sin parámetros.
- Se pueden definir en una clase más de un constructor.
- Se diferencian entre sí por la cantidad u orden de los parámetros (sobrecarga de métodos).
- Se ejecutan luego de utilizar el operador new.

Ejemplo:

En la misma clase Auto que definimos los atributos anteriores.

```
public class Auto {  
    .....  
  
    //Constructor sin parámetros  
    public Auto(){  
        this.Matricula="Sin matricula";// Usamos la property Matricula  
        // Se podría también asignar el valor de la matrícula utilizando directamente  
        // el atributo: this.matricula = "Sin matricula"  
        _marca="Sin marca";  
        _modelo="Sin asignar";  
    }  
  
    //Constructor con parámetros e inicialización de los mismos  
    public Auto(string matricula,string marca, string modelo, int añoFabricacion){  
        _matricula=matricula;  
        _marca=marca;  
        _modelo=modelo;  
        _añoFabricación=añoFabricacion;  
    }  
}
```

Constructor – Crear instancia

- Por ejemplo, en el método Main de la clase Program (aplicación de consola), para crear (instanciar) un objeto Auto escribiremos:

//utilizando constructor sin parámetros o por defecto

```
Auto unAuto = new Auto();
```

// o utilizando el constructor con parámetros

```
Auto unAuto = new Auto("SAP1234","Ford","Fiesta",2014);
```

- Si no definimos ningún constructor en la clase, se invoca al constructor por defecto, que inicializa los atributos de la clase con los valores por defecto según el tipo de dato de sus atributos.

ToString()

- Método que retorna una cadena de texto con los datos que se desean mostrar de los objetos de esa clase.
- Si el método es llamado y no está declarado en la clase, compila pero muestra una dirección de memoria y no los datos del Objeto.

Ejemplo

En la clase Auto agregamos:

```
public override string ToString(){  
    return this.marca + this.matricula + this.modelo;  
}
```

ToString() - Pruebas

Sin definir el ToString() en la clase Auto

```
public class Program
{
    0 referencias
    public static void Main(string[] args)
    {
        Auto unAuto = new Auto();
        Console.WriteLine(unAuto);
        Console.ReadKey();
    }
}
```

En la consola aparece:



Pruebas.Auto

ToString() - Pruebas

Con el ToString() definido en la clase Auto

```
public override string ToString()
{
    return this.Marca + " - " + this.Matricula + " - " + this.Modelo;
}
```

En Program usando constructor sin parámetros:

```
public class Program
{
    // Referencias
    public static void Main(string[] args)
    {
        Auto unAuto = new Auto();
        Console.WriteLine(unAuto);
        Console.ReadKey();
    }
}
```

En la consola aparece:

```
Sin marca - Sin matricula - Sin asignar
```

ToString() - Pruebas

Con el ToString() definido en la clase Auto

```
public override string ToString()
{
    return this.Marca + " - " + this.Matricula + " - " + this.Modelo;
}
```

En Program usando constructor con parámetros:

```
public class Program
{
    Oreferencias
    public static void Main(string[] args)
    {
        Auto unAuto = new Auto("SAP1234", "Ford", "Fiesta", 2014);
        Console.WriteLine(unAuto);
        Console.ReadKey();
    }
}
```



En la consola aparece:

```
Ford - SAP1234 - Fiesta
```

¿Por qué no muestra el año de fabricación ?

Atributos y métodos

Instancia: se pueden usar desde la instancia del objeto (objeto creado); un objeto llama a un método para realizar una determinada tarea.

Clase: solo se pueden usar con miembros de clase; el método se llama desde la propia clase, se invocan con el **NombreClase.NombreMetodo**; y no requieren de una instancia del objeto para poder ser invocados

Los atributos y métodos de clase se declaran con la palabra `static` al igual que los atributos.

Por ejemplo, se puede usar para validar datos antes de instanciar el nuevo objeto.

Ejemplo método de clase:

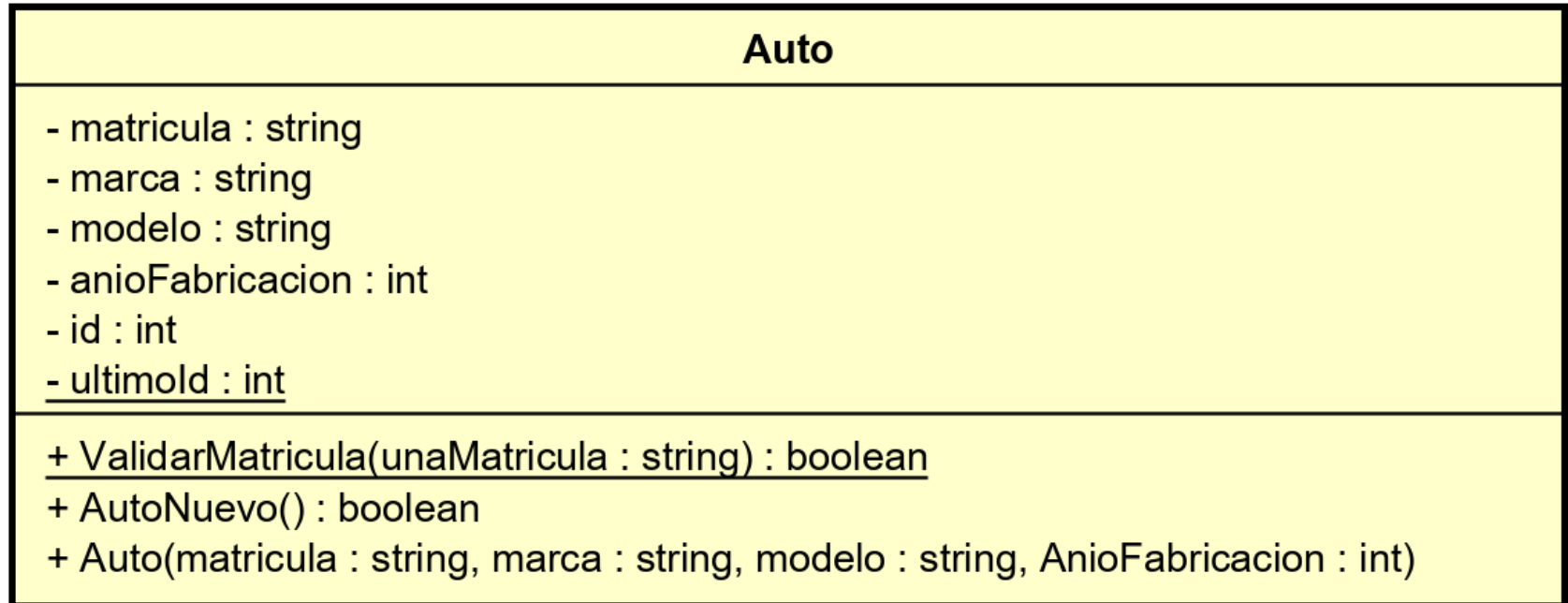
//declaración del método

```
public static string ValidarMatricula(string matricula){  
}
```

//llamada al método

```
Auto.ValidarMatricula(matricula);
```


Diagrama Uml



Método de clase

En clase Auto:

```
public static Boolean ValidarMatricula(string unaMatricula)
{
    return unaMatricula.Length == 7;
}
```

En clase Auto:

```
public class Program
{
    Referencias
    public static void Main(string[] args)
    {
        if (Auto.ValidarMatricula("SAP123"))
        {
            Auto unAuto = new Auto("SAP123", "Ford", "Fiesta", 2014);
            Console.WriteLine(unAuto);
        }
        else
        {
            Console.WriteLine("La matricula no es correcta");
        }
        Console.ReadKey();
    }
}
```

Auto
<ul style="list-style-type: none"> - matricula : string - marca : string - modelo : string - anioFabricacion : int - id : int - ultimold : int
<ul style="list-style-type: none"> + ValidarMatricula(unaMatricula : string) : boolean + AutoNuevo() : boolean + Auto(matricula : string, marca : string, modelo : string, AnioFabricacion : int)

En la consola aparece:

```
La matricula no es correcta
```

Método de instancia

En clase Auto:

```
public bool AutoNuevo()
{
    return matricula == "";
}
```

En clase Auto:

```
static void Main(string[] args)
{
    Auto unAuto = new Auto("SAP123", "Ford", "Fiesta", 2014);
    if (unAuto.AutoNuevo())
    {
        Console.WriteLine("El auto nuevo");
    }
    Console.ReadKey();
}
```

Auto
- matricula : string - marca : string - modelo : string - anioFabricacion : int - id : int - ultimoId : int
+ ValidarMatricula(unaMatricula : string) : boolean + AutoNuevo() : boolean + Auto(matricula : string, marca : string, modelo : string, AnioFabricacion : int)

Equals

El método Equals es un método que retorna un dato bool que permite establecer si un objeto es igual a otro.

Por defecto el método Equals definido en object compara referencias para saber si el objeto es el mismo.

Ejemplo:

```
clase Program
```

```
static void main(string [] args){
```

```
    Alumno a = new Alumno();
```

```
    Alumno b = a;
```

```
    if(a.Equals(b)){ //retorna true ambas variables referencian al mismo objeto}}
```

Equals

Otro ejemplo:

clase Program

```
public static void main(string [] args){  
    Alumno a = new Alumno();  
    Alumno b = new Alumno();  
    If(a.Equals(b)){ //retorna false; las referencias son distintas  
}
```

Si se desea se puede sobrescribir el método Equals en la clase correspondiente indicando que valor se desea indicar como único en el objeto; es decir no deberían de poder existir dos objetos con el mismo valor en ese atributo en particular.

Equals

Si los objetos alumnos tienen un numEstudiante que no se puede repetir podríamos sobrecribir el método Equals para que retorne true si el numEstudiante es el mismo.

Para ello debemos de escribir en la clase Alumno lo siguiente:

```
public override bool Equals(Object obj){  
    if (obj == null || !(obj is Alumno))  
        return false;  
    else  
        return this.NumEstudiante==((Alumno)obj).NumEstudiante;  
}
```

GetHashCode

Un código hash **es** un valor numérico que se utiliza **para** identificar un objeto durante las pruebas **de** igualdad.

Ejemplo:

```
Private int id;  
  
public override int GetHashCode()  
{  
    return id;  
}
```