# Clases

* Definición de un tipo de objeto que indica sus atributos y métodos.
* Molde o plantilla a partir de la cual se crean objetos.
* Diseñadas con una responsabilidad en particular (cohesión).
* Pueden depender en cierto grado de otras clases (acoplamiento).

Una clase debe tener un único objetivo, si tiene múltiples objetivos va a perder cohesión y va a ser un programa más difícil de controlar.

Mientras menos acoplamiento tengan las clases, un mejor software vamos a tener.

Para crear una clase, utilice la palabra reservada **class** y enseguida el nombre de la clase comenzando con la primera letra en mayúscula.

namespace Clases

{

class CuentaBancaria

{

}

}

## Modificadores de acceso

La palabra reservada public es un modificador de acceso, que se utiliza para establecer el nivel de acceso/visibilidad para clases, campos, métodos y propiedades.

C# tiene los siguientes modificadores de acceso:

**public:** El código es accesible para todas las clases, no tiene restricciones.

**private:** El código solo es accesible dentro de la misma clase.

**protected:** El código es accesible dentro de la misma clase, o en una clase que se hereda de esa clase.

**internal:** El código solo es accesible dentro de su propio assembly, pero no desde otro assembly.

También hay dos combinaciones**: protected internal** y **private protected**.

**protected internal:** El acceso está limitado al assembly o tipos actuales derivados de la clase contenedora.

**private protected:** El acceso está limitado a la clase o tipos que contienen derivados de la clase contenedora dentro del conjunto actual. Disponible desde C# 7.2.

Solo se permite un modificador de acceso para un miembro o tipo, excepto cuando usa las combinaciones protegidas internas o privadas protegidas.

Los modificadores de acceso no están permitidos en los espacios de nombres (namespaces). Los espacios de nombres no tienen restricciones de acceso.

## Constructores

Un constructor es un método especial que se utiliza para inicializar objetos. La ventaja de un constructor, es que se llama cuando se crea un objeto de una clase. Se puede utilizar para establecer valores iniciales para los campos. Una clase puede tener varios constructores.

Cada clase recibe un constructor por default, sí y solo sí no tiene definido algún otro constructor, es decir, desaparece el constructor automático cuando se define algún constructor.

La diferencia entre un método y un constructor es que el constructor no tiene tipo de retorno.

El constructor debe llamarse igual que la clase.

Ejemplos:

//Constructor vacío por default

public CuentaBancaria()

{

}

// se le piden parámetros al momento de instanciar una clase para asignarlos al atributo correspondiente.

public CuentaBancaria(string noCuenta)

{

NoCuenta = noCuenta;

}

// constructor que inicializa todas las variables

public CuentaBancaria(string noCuenta, string usuario, decimal saldo)

{

NoCuenta = noCuenta;

Usuario = usuario;

Saldo = saldo;

}

### Evitar código repetitivo en constructores

Un constructor puede llamar a otro. Primero llama al constructor que recibe los datos, asigna los valores y después se regresa al constructor y asigna el último dato, así de esta manera evitamos tener código repetido y se sigue un patrón orientado a objetos.

//Constructor vacío o por default

public CuentaBancaria() { }

public CuentaBancaria(string noCuenta)

{

NoCuenta = noCuenta;

}

public CuentaBancaria(string noCuenta, string usuario) : this(noCuenta)

{

Usuario = usuario;

}

public CuentaBancaria(string noCuenta, string usuario, decimal saldo) : this(noCuenta, usuario)

{

Saldo = saldo;

}

## Atributos

Los atributos también llamados propiedades son valores que pertenecen a una clase y que nos van a servir para guardar un tipo de información.

Existen dos tipos de propiedades: propiedades automáticas y propiedades completas.

Propiedades automáticas: son las podemos utilizar para leer y escribir datos en ellas directamente sin tener que escribir ningún código adicional.

// Auto properties (propiedades automáticas)

public string NoCuenta { get; set; }

public string Usuario { get; set; }

public decimal Saldo { get; set; }

Propiedades completas o Full properties: La principal utilidad de las full properties es que podemos hacer uso de los métodos especiales get y set para efectuar algunas validaciones.

Lo primero que necesitamos es una variable que nos sirva para guardar el valor de esa propiedad. A eso se le llama backing field, es un campo en la clase que nos va a permitir guardarlo. Este campo va a ser privado y la convención para este tipo de variables es que comiencen con un guión bajo.

// backing field

private decimal \_saldo;

Después de tener el campo, podemos crear la propiedad completa.

Comenzamos escribiendo public, después el tipo, luego el nombre y abrimos y cerramos llaves. Dentro de las llaves pondremos los métodos get y set extendidos.

// Full property (propiedad completa)

public decimal Saldo

{

get { return \_saldo; }

set {

// Evitar que el saldo sea negativo

if( value < 0)

{

\_saldo = 0;

}

else

{

\_saldo = value;

}

}

}

## Métodos

Un método es un bloque de código que sólo se ejecuta cuando se llama.

Pueden pasársele datos, conocidos como parámetros, a un método.

Los métodos se utilizan para realizar ciertas acciones, y también se conocen como funciones.

void significa que este método no tiene un valor devuelto.

// métodos

public void Retirar(decimal cantidad)

{

if ( cantidad > Saldo )

{

Console.WriteLine("Saldo insuficiente");

// Evitar que el código se siga ejecutando

return; // sale del método en este punto

}

Saldo = Saldo - cantidad;

}

public void Depositar(decimal cantidad)

{

Saldo += cantidad;

}

# Objetos

* Entidad con estado y comportamiento.
* Estructura de datos (color, modelo, marca) junto con sus operaciones (acelerar, frenar, girar).
* Instancia de una clase.

Su comportamiento son las acciones que puede realizar.

Su estado son las propiedades o entidades que lo hacen único.

## Instancia de clases

// Crear un objeto de tipo CuentaBancaria

CuentaBancaria cuenta = new CuentaBancaria();

El término Sobrecarga es cuando varios métodos pueden tener el mismo nombre con diferentes parámetros.

Nuestra clase puede ser instanciada con diferentes parámetros al tener sobrecarga en los constructores.

// Usando el constructor con dos argumentos

CuentaBancaria cuenta2 = new CuentaBancaria("000002", "John Doe");

Podemos instanciar e inicializar variables de una manera más sencilla pasándole los parámetros entre llaves.

// Sintaxis de inicialización

CuentaBancaria cuenta3 = new CuentaBancaria { NoCuenta = "000003", Usuario = "Jane Doe", Saldo = 10000 };

## Invocar propiedades o métodos de objetos

Podemos acceder a las propiedades y asignarles valores a las propiedades de los objetos.

cuenta.NoCuenta = "000001";

cuenta.Usuario = "Joseph Doe";

cuenta.Saldo = 10000;

Llamar, invocar o ejecutar un método se usan como sinónimos.

También podemos invocar los métodos del objeto y podemos guardar la respuesta en una variable en caso de que devuelva algo.

cuenta.Depositar(50);

cuenta.Retirar(20);

# Bibliografía

<https://www.w3schools.com/cs/cs_classes.php>

<https://www.w3schools.com/cs/cs_access_modifiers.php>

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/accessibility-levels>

<https://www.w3schools.com/cs/cs_constructors.php>

<https://www.w3schools.com/cs/cs_properties.php>

<https://www.udemy.com/course/c-sharp-para-principiantes/learn/lecture/4801520>