# Ejercicio CSharp Modelo y Servicio 2

Supongamos la siguiente clase de modelo

|  |
| --- |
| class Cliente {      // campos      public string nombre;      public string apellido;      public int saldo;  } |

Mientras esta bien, existe una restricción de OOP (Programación orientada a objeto) que dice:

* Los campos deben ser privados.

Existen dos alternativas para trabajar esto.

## crear constructor

El constructor se define como un método con el mismo nombre de la clase.

El constructor no devuelve dato (ni tampoco se debe indicar void).

Y el constructor sirve para inicializar los datos cuando se crear el objeto.

|  |
| --- |
| class Cliente {      // campos      private string nombre;      private string apellido;      private int saldo;      public Cliente(string nombre, string apellido,int saldo) {          this.nombre = nombre;          this.apellido = apellido;          this.saldo = saldo;      }  }  // y se usa como  var cliente=new Cliente("john","doe",0); |

El constructor se puede crear automáticamente:

1. Botón derecho dentro de la clase (en cualquier parte vacía)
2. Quick actions
3. Constructor
4. Y seleccione los campos que quiere agregar.

Nota: Es conveniente agregar dos constructores, uno con todos los campos y uno sin campos. Para crear ambos constructores, repita estos pasos dos veces.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

## agregar propiedades

Otra alternativa es usar propiedades. Las propiedades funcionan como métodos set/get y a la vez como campos.

|  |
| --- |
| class Cliente {      // propiedades      public string Nombre {set; get};      public string Apellido {set; get};      public int Saldo {set; get};  }  // y se usa como  var cliente=new Cliente();  cliente.Nombre="john";  cliente.Apellido="doe";  cliente.Saldo=0; |

|  |  |
| --- | --- |
| Burbuja de chat | Las propiedades se entienden como métodos, por lo tanto, se recomienda iniciar con la letra mayúscula. |

## Ejercicio 1

Cree un proyecto nuevo de consola, y dentro de él, cree las siguientes clases, asígnele propiedades y dos constructores (uno vacio y uno con todos los atributos) a cada una de las clases.

* CuentaBanco
  + Int numero
  + String propietario
  + Int balance
* Transaccion
  + Int cantidad
  + String fecha // la fecha como texto
  + String descripción

Y en program.cs, cree dos variables del tipo **CuentaBanco** y **Transaccion**, y muestre los datos en la consola.

## Clases de servicio

Una clase de servicio es una clase con una colección de funciones. Usualmente, una clase de servicio no tiene campos.

Ejemplo:

|  |
| --- |
| class ClienteServicio {      public void agregarDeuda(Cliente clim,int monto) {          clim.saldo=clim.saldo-monto;      }      public void agregarDeposito(Cliente clim,int monto) {          clim.saldo=clim.saldo+monto;      }      public void mostrar(Cliente clim) {          Console.WriteLine($"{clim.nombre},{clim.apellido},{clim.saldo}");      }  } |

Para usar esta clase y acceder a la clase se usa el siguiente código.

|  |
| --- |
| var cliSrv=new ClienteServicio();  cliSrv.agregarDeposito(cli,200);  cliSrv.agregarDeuda(cli,50);  cliSrv.mostrar(cli); |

El problema de este método, es que hay que crear una variable del tipo ClienteServicio.

Hay 3 alternativas para resolver este problema: static, singleton e inyección de dependencia. En este caso veremos estatico:

Para crear una clase de servicio con métodos estáticos, se agrega el comando “static” en cada método.

Ejemplo:

|  |
| --- |
| class ClienteServicio {      public static void agregarDeuda(Cliente clim,int monto) {          clim.saldo=clim.saldo-monto;      }      public static void agregarDeposito(Cliente clim,int monto) {          clim.saldo=clim.saldo+monto;      }      public static void mostrar(Cliente clim) {          Console.WriteLine($"{clim.nombre},{clim.apellido},{clim.saldo}");      }  } |

|  |  |
| --- | --- |
| Advertencia con relleno sólido | **Advertencia:**  Métodos estaticos no deben usarse si la clase de servicio tiene estados (campos). |

## Ejercicio 2

Usando el proyecto anterior, cree las siguientes clases de servicio

* **CuentaBancoServicio**
  + Esta función créela con métodos estáticos
  + Con una función que agregue un balance a una cuenta (vea en el ejemplo agregarDeposito)
  + Otra función que quite dinero del balance de la cuenta (vea en el ejemplo agregarDeuda)
  + Y otra función que muestre la cuenta
  + Y use la cuenta en program.cs
* **TransaccionServicio**
  + Esta función créela con métodos estáticos
  + Y agregue una función para mostrar una transacción.