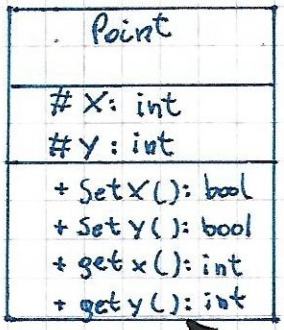
Clases y Objetos
(1) (Herencia)

- **Herencia Simple**

Utilizamos herencia cuando definimos una clase utilizando otra clase como punto de partida.



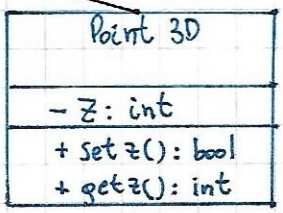
Nota: los miembros de datos públicos y protegidos; y los métodos públicos y protegidos de la clase base Point, están disponibles también para la clase derivada Point3D.

Si hubiese un atributo o método privado, el mismo no estaría disponible en la clase derivada.

Importante: C# no permite herencia múltiple (una clase que deriva de varias clases base.

Una clase derivada se define así:

```
Pubic class Point3D : Point
{
   ....
   ....
}
```



- **Cómo redefinir comportamiento en una clase derivada**

```
Public class ClaseBase
{
  public virtual void MostrarMensaje()
  {
    Console.WriteLine ("Método invocado desde la ClaseBase");
  }
}

Public class ClaseHeredada: ClaseBase
{
  public override void MostrarMensaje()
  {
    Console.WriteLine ("Método invocado desde la ClaseHeredada");
  }
}
```

Con "virtual" hacemos que dicho método pueda ser redefinido en las clases hijas.

Con "override" podemos redefinir el comportamiento del método. Si no usamos "override" tendremos un error de compilación.

En el main:

```
ClaseBase objeto1 = new ClaseBase();
ClaseHeredada objeto2 = new ClaseHeredada();
objeto1.MostrarMensaje(); //"Método invocado desde la clase Base"
objeto2.MostrarMensaje(); //"Método invocado desde la ClaseHeredada"
```

• Clase Abstracta

Una clase definida como "Abstract" no puede instanciarse y tiene las siguientes características:

1 – Puede declarar métodos y propiedades abstractos (clases derivadas los implementarán).
2 – Puede declarar métodos no abstractos (codificando su implementación).
3 – Solo puede heredar a otra clase abstracta o a una clase que implemente el método o los métodos abstractos definidos en la superclase.
4 – Sólo Puede declarar Atributos no abstractos. También Propiedades no abstractas y abstractas.
5 – Su utilidad consiste en proveer estructura y comportamiento común a todas las subclases que heredan de ella.

```
public abstract class ClaseBase
{
    public int atributo;          → Un atributo sólo se puede declarar como no abstracto

    public int propiedad          → Las propiedades pueden ser abstractas o no abstractas,
    {                                 si lo son, deberán implementarse en las clases derivadas
        get;
        set;
    }

    public abstract void MostrarMsj();   → Los métodos pueden declararse como abstractos
}                                          o no abstractos. Si son abstractos, todas las
                                           clases derivadas deberán implementarlos.

public class Clase1 : ClaseBase          → Implementamos el método abstracto con "override"
{
    public override void MostrarMsj()
    {
        Console.WriteLine("Invocado desde Clase1");
        atributo = 8;   → Podemos usar los atributos no abstractos declarados en la clase abstracta.
    }
}
```

Si quisiéramos declarar, en la clase abstracta, un método no abstracto pero que su comportamiento pueda ser redefinido en las clases derivadas, podemos hacerlo así:

```
public virtual void MetodoX()   → La palabra virtual nos permitirá escribir otra
{                                  implementación para el método en sus
    ...                            clases derivadas
    ...   → Como no es un método
}           abstracto, es obligatorio
            escribir una implementación
            básica.
```

(2) Clases y Objetos (Aclarando Conceptos)

Los tipos en C#.pdf Pág 26 y 27

- **Variables de instancia**

Cualquier campo o atributo declarado en una clase sin el modificador static, se considera una variable de instancia.

```
Class Ejemplo
{
    public int x;
    ...
    ...
}
```

- Una variable de instancia puede pertenecer a una clase o a un "struct".
- Cada objeto tendrá sus propias variables de instancia

Su uso podría ser:

```
Ejemplo e1 = new Ejemplo();
Ejemplo e2 = new Ejemplo();

e1.x = 10;
e2.x = 20;
```

En este ejemplo, la variable x de la instancia e1 valdrá 10 mientras que la de la instancia e2 valdrá 20

- **Variables de clase o "static"**

Las características principales de las variables static son:

1) No es necesario crear una instancia de la clase para poder acceder a ellas.
2) Sólo hay una para todas las instancias de la clase que la contiene (de forma contraria a las variables de instancia, donde hay una por cada objeto creado).

```
Class Ejemplo
{
    public static int x;
    ...
    ...
}
```

Su uso podría ser

```
Ejemplo.x = 5; // esto es válido

Ejemplo e1 = new Ejemplo(),
Ejemplo e2 = new Ejemplo();

e1.x = 10;
```

↳ esta asignación es inválida para el compilador.

• Métodos static

Un método estático puede invocarse sin tener que crear un objeto de la clase que lo contiene. Un método estático tiene algunas restricciones:

1) No puede acceder a los atributos de la clase (salvo que sean estáticas).
2) No puede usar el operador "this" ya que el mismo haría referencia a un objeto inexistente
3) Puede invocar a otros métodos siempre y cuando estos tambien sean estáticos.

Un ejemplo común de método estático es:

```
static void Main (string [] args)
```

Otro ejemplo:

```
public static int Multiplicar (int a, int b)
```

Nota: la palabra static en la declaración del método, significa que el compilador sólo debe permitir que exista en memoria, una copia del método por vez. Como el método Main() es el punto de entrada a la aplicación, sería catastrófico permitir que el punto de entrada se abriese más de una vez..." Biblia de C# (pág 54).

Ejercicio: Implementar una clase llamada "Operacion". Definir dos métodos estáticos que permitan sumar y restar dos valores enteros (ingresados por teclado).

Un método estático se invoca anteponiendo el nombre de la clase que lo contiene, por ejemplo: `Operacion.Multiplicar (2,3);`

Clase siguiente

• Clases Estáticas

Una clase estática difiere de una clase convencional en que no se pueden crear instancias de la misma. En otras palabras, no puede utilizarse la palabra clave "new" para crear objetos instancia de la clase.

```
static class EmpleadoEmpresa
{
    public static void HacerAlgo ()
    {
    ...
    }
    public static void HacerOtraCosa ()
    {
    ...
    }
}
```

→ si la clase es estática, todos sus miembros internos deben ser estáticos.

- Una clase estática puede tener un constructor, pero el mismo debe ser estático también.

Clases y Objetos
(3) (Aclarando conceptos)

Crear una clase estática es, por consiguiente, básicamente igual que crear una clase que sólo contiene miembros estáticos y un constructor privado.*1 La ventaja de usar una clase estática, es que el compilador garantizará que no se puedan crear instancias de la clase.

Ejemplo de una clase no estática que contiene únicamente miembros estáticos pero que no está marcada como estática:

```
public class Empleado
{
    static Empleado ()
    {
      ... // código del constructor
    }
    public static void HacerAlgo ()
    {
      ...
    }
}
```

OBSERVACIONES:

Al igual que una clase estática, esta clase permite acceder a uno de sus miembros directamente:

```
Empleado.HacerAlgo();
```

Que el constructor sea Estático hace que el mismo pueda invocarse al crear un objeto:

```
Empleado ee = new Empleado();
```

Pero si instanciamos, no podremos acceder a los miembros estáticos:

```
ee.HacerAlgo(); → Error de compilación
```

Si no establecemos esta clase como estática, podemos crear miembros de instancia (objetos) de forma accidental.

*1: Si bien es posible crear una clase que sólo tenga miembros estáticos excepto su constructor que debería ser privado, es mucho más sencillo establecer la clase como static con lo que logramos el mismo cometido (y no necesitamos escribir un constructor).

```
public static class Empleado
{
    public static void HacerAlgo ()
    {
      ...
    }
}
```

son similares

```
public class Empleado
{
    private Empleado ()
    {
      ...
    }
    public static void HacerAlgo ()
    {
      ...
    }
}
```

El constructor privado evita que se creen instancias de la clase

- Una clase estática podría servir para definir una serie de métodos genéricos a modo de "librería". Un buen ejemplo es la clase "Math".
- Las clases estáticas son de tipo "sealed", y por consiguiente no pueden heredarse. Tampoco pueden heredar de cualquier clase (excepto Object).

- Clases selladas (sealed)

```
public sealed class ClaseSellada
{
  ...
}
```

A través de una clase sellada restringimos el ámbito de herencia, es decir, definimos una clase que no se puede heredar.