

PASO A PASO MÓDULO 9

Creación, uso de clases y destrucción de objetos

Objetivos

Al final de esta práctica, usted será capaz de:

- Acceder a datos según sus modificadores.
- Usar herencia entre clases.

Requisitos previos

Antes de realizar la práctica debe estar familiarizado con los siguientes temas:

- Crear clases y hacer instancias de objetos.
- Paso de argumentos como parámetros de métodos en C#

EJERCICIO 1

En este ejercicio veremos la estructura de herencia entre diferentes objetos de transporte, y la interrelación entre sus miembros.

Paso 1. Crea un nuevo Proyecto en Visual C# de tipo Aplicación de Consola, de nombre Vehiculos.

Paso 2. Crea la clase vehículo, que contendrá los atributos (variables) siguientes: Color, Caballos, Marca, Modelo

Decide tú mismo el tipo de dato de cada variable, y hazlas todas "public" para que sean accesibles desde fuera de la clase.

Crea un constructor por defecto y otro que tenga por parámetros todos los anteriores atributos.

```
class Vehiculo
{
    public string Color;
    public int Caballos;
    public string Marca;
    public string Modelo;
    public Vehiculo()
    {
        this.Color = "";
        this.Caballos = 0;
        this.Marca = "";
        this.Modelo = "";
    }
    public Vehiculo(string colo, int cab, string mar, string mod)
    {
        this.Color = colo;
        this.Caballos = cab;
        this.Marca = mar;
        this.Modelo = mod;
    }
}
```

Paso 3. Realiza un programa que instancie un objeto de la clase Vehículo mediante el constructor por defecto. A continuación asigna un valor a cada variable de este objeto.

```
static void Main(string[] args)
{
    Vehiculo v = new Vehiculo();
    v.Color = "amarillo";
    v.Caballos = 180;
    v.Marca = "Nissan";
    v.Modelo = "Primera";
}
```

Paso 4. Modifica la clase Vehículo de la siguiente forma: cambia los “public” de las variables por “private”. Después intenta ejecutar el Main del ejercicio anterior. ¿Qué sucede? ¿Y si usamos “protected”, o nada? Explica el porqué del resultado en cada caso.

```
class Vehiculo
{
    private string Color;
    private int Caballos;
    private string Marca;
    private string Modelo;
    public Vehiculo()
    {
        this.Color = "";
        this.Caballos = 0;
        this.Marca = "";
        this.Modelo = "";
    }
    public Vehiculo(string colo, int cab, string mar, string mod)
    {
        this.Color = colo;
        this.Caballos = cab;
        this.Marca = mar;
        this.Modelo = mod;
    }
}
```

```
static void Main(string[] args)
{
    Vehiculo v = new Vehiculo();
    v.Color = "amarillo";
    v.Caballos = 180;
    v.Marca = "Nissan";
    v.Modelo = "Primera";
}
```

string Vehiculo.Modelo

Error:

'Construcciones.Program.Vehiculo.Modelo' no es accesible debido a su nivel de protección

Public: Visible para todo el mundo.

Private: Visible a la clase solo. Aquí da error ya que las variables están sin acceder a ellas.

Protected: Visible para el paquete y todas las subclases.

Nada: El por defecto. Visible para el paquete.

Paso 5. Resuelve el problema presentado anteriormente sin introducir código nuevo en la clase Vehiculo, utilizando el constructor de 4 parámetros:

```
static void Main(string[] args)
{
    Vehiculo v = new Vehiculo("amarillo",180, "Nissan","Primera");
}
```

Paso 6. Crea una clase Coche, que heredará de Vehículo, y además contendrá las siguientes variables: numPuertas y capacidadMaletero

Nota: En este caso sin ningún modificador.

```
class Coche : Vehiculo
{
    int numPuertas;
    int capacidadMaletero;
}
```

Paso 7. Modifica el Main de manera que se añada un objeto de tipo Coche. Dale valor a numPuertas y capacidadMaletero. ¿Funciona? ¿A que tipo de nivel de protección corresponde la ausencia de modificador?

```
static void Main(string[] args)
{
    Vehiculo v = new Vehiculo("amarillo",180, "Nissan","Primera");
    Coche co = new Coche();
    co.numPuertas=0;
    co.capacidadMaletero = 0;
}
```

```
int Coche.capacidadMaletero
```

Error:

'Construcciones.Program.Coche.capacidadMaletero' no es accesible debido a su nivel de protección

En Java la ausencia de modificador coincide con public. En C# se corresponde con private

Paso 8. Crea un constructor de Coche por defecto (con ningún parámetro) y otro con 6 parámetros, de forma que los 4 primeros inicialicen los atributos heredados de la clase Vehículo y los dos últimos hagan lo propio con los datos que aporta esta clase.

```
class Coche : Vehiculo
{
    public int numPuertas;
    public int capacidadMaletero;
    public Coche(string colo, int cab, string mar, string mod,
        int npuertas,int maletero) : base(colo, cab, mar, mod)
    {
        this.numPuertas = npuertas;
        this.capacidadMaletero = maletero;
    }
    public Coche(): base()
    {
        this.numPuertas = 0;
        this.capacidadMaletero = 0;
    }
}
```

Base() llamó al constructor sin parámetros de Vehículo que inicializará sus atributos

Paso 9. Corrige el error del paso 9, utilizando el nuevo constructor de la clase Coche, que permitirá inicializar los atributos heredados de la clase padre y los propios:

```
static void Main(string[] args)
{
    Vehiculo v = new Vehiculo("amarillo",180, "Nissan","Primera");
    Coche co = new Coche("rojo",200,"Nissan","Almera",0,0);
}
```

Paso 10. Haz la clase Vehículo "abstract". ¿Qué sucede al ejecutar el main? ¿Por qué? Además, explica qué utilidad tiene.

```
abstract class Vehiculo
{
    private string Color;
    private int Caballos;
    private string Marca;
    private string Modelo;
    public Vehiculo()
    {
        this.Color = "";
        this.Caballos = 0;
        this.Marca = "";
        this.Modelo = "";
    }
    public Vehiculo(string colo, int cab, string mar, string mod)
    {
        this.Color = colo;
        this.Caballos = cab;
        this.Marca = mar;
        this.Modelo = mod;
    }
}
```

```
static void Main(string[] args)
{
    Vehiculo v = new Vehiculo("amarillo",180, "Nissan","Primera");
    Coche co = new Coche("rojo",200,"Nissan","Almera",0,0);
}
```

Se produce un error, porque no se pueden crear instancias de una clase abstracta. El propósito de una clase abstracta es proporcionar una definición común de una clase base que las clases derivadas pueden compartir.

Por ejemplo, una biblioteca de clases puede definir una clase abstracta que se utiliza como parámetro para muchas de sus funciones y solicitar a los programadores que utilizan esa biblioteca que proporcionen su propia implementación de la clase mediante la creación de una clase derivada.

Las clases abstractas también pueden definir métodos abstractos. Esto se consigue agregando la palabra clave `abstract` antes del tipo de valor que devuelve el método. Los métodos abstractos no tienen ninguna implementación, de modo que la definición de método va seguida por un punto y coma en lugar de un bloque de método normal. Las clases derivadas de la clase abstracta deben implementar todos los métodos abstractos. Cuando una clase abstracta

hereda un método virtual de una clase base, la clase abstracta puede reemplazar el método virtual con un método abstracto. Por ejemplo:

```
public class D
{
    public virtual void DoWork(int i)
    {
        // Original implementation.
    }
}

public abstract class E : D
{
    public abstract override void DoWork(int i);
}

public class F : E
{
    public override void DoWork(int i)
    {
        // New implementation.
    }
}
```

Si un método virtual se declara como abstract, sigue siendo virtual para cualquier clase que herede de la clase abstracta. Una clase que hereda un método abstracto no puede tener acceso a la implementación original del método; en el ejemplo anterior, DoWork de la clase F no puede llamar a DoWork de la clase D. De esta forma, una clase abstracta puede obligar a las clases derivadas a proporcionar nuevas implementaciones de método para los métodos virtuales.

Paso 11. Ahora haz la clase Vehículo “sealed”. ¿Qué sucede al ejecutar el main? ¿Por qué? Además, explica qué utilidad tiene.

```
class Coche : Vehiculo
{
    in 'Construcciones.Program.Coche': no puede derivar del tipo sealed 'Construcciones.Program.Vehiculo'
    int capacidadMaletero;
    public Coche(string colo, int cab, string mar, string mod,
        int npuertas, int maletero)
        : base(colo, cab, mar, mod)
    {
        this.numPuertas = npuertas;
        this.capacidadMaletero = maletero;
    }
}
```

```
sealed class Vehiculo
{
    private string Color;
    private int Caballos;
    private string Marca;
    private string Modelo;
    public Vehiculo()
    {
        this.Color = "";
        this.Caballos = 0;
        this.Marca = "";
        this.Modelo = "";
    }
    public Vehiculo(string colo, int cab, string mar, string mod)
    {
        this.Color = colo;
        this.Caballos = cab;
        this.Marca = mar;
        this.Modelo = mod;
    }
}
```

Una clase sealed no se puede utilizar como clase base. Por esta razón, tampoco puede ser una clase abstracta. Las clases selladas evitan la derivación. Puesto que nunca se pueden utilizar como una clase base, algunas optimizaciones en tiempo de ejecución pueden hacer que sea un poco más rápido llamar a miembros de clase sellada.

Un miembro de clase, método, campo, propiedad o evento de una clase derivada que reemplaza a un miembro virtual de la clase base puede declarar ese miembro como sellado. Esto niega el aspecto virtual del miembro para cualquier clase derivada adicional. Esto se logra colocando la palabra clave sealed antes de la palabra clave override en la declaración del miembro de clase. Por ejemplo:

```
public class D : C
{
    public sealed override void DoWork() { }
}
```

Paso 12. Deja la clase Vehiculo normal (eliminando la clave abstract o sealed). A continuación declara los atributos de la clase Coche con la clave readonly de la siguiente forma:

```
class Coche : Vehiculo
{
    public readonly int numPuertas;
    public readonly int capacidadMaletero;
    public Coche(string colo, int cab, string mar, string mod,
        int npuertas, int maletero) : base(colo, cab, mar, mod) {
        this.numPuertas = npuertas;
        this.capacidadMaletero = maletero;
    }
    public Coche(): base() {
        this.numPuertas = 0;
        this.capacidadMaletero = 0;
    }
}
```

Paso 13. Intenta acceder a una de las variables públicas del objeto coche y modifica su valor. ¿Qué sucede cuando se intenta asignar un valor a una de estas variables desde Main?

```
static void Main(string[] args)
{
    Vehiculo v = new Vehiculo("amarillo",180, "Nissan","Primera");
    Coche co = new Coche("rojo",200,"Nissan","Almera",0,0);
    co.capacidadMaletero = 6;
}

```

No se puede asignar un campo de sólo lectura (excepto en un constructor o inicializador de variable)

readonly declara las variables como constantes (sólo pueden ser inicializadas una vez) y por lo tanto hace que el valor no pueda variar durante la ejecución del programa.

Final en clases: Para evitar que herencia de clases

Java

```
public final class MyFinalClass {...}

```

C#

```
public sealed class MyFinalClass {...}

```

Final en métodos: Para evitar que sobrescriban los métodos en herencia

Java

```
public class MyClass
{
    public final void myFinalMethod() {...}
}

```

C#

```
public class MyClass : MyBaseClass
{
    public sealed override void MyFinalMethod() {...}
}

```

Final en atributos: Para evitar que sean asignados más de una vez.

Java

```
public final double pi = 3.14; // essentially a constant

```

C#

```
public readonly double pi = 3.14; // essentially a constant

```

Paso 14. Ahora en la clase Coche declara sus atributos con la clave static. ¿Que pasa en el constructor?

```

class Coche : Vehiculo
{
    public static int numPuertas;
    public static int capacidadMaletero;
    public Coche(string colo, int cab, string mar, string mod,
        int npuertas,int maletero) : base(colo, cab, mar, mod) {
        this.numPuertas = npuertas;
        this.capacidadMaletero = maletero;
    }
    public Coche(): base() {
        this.numPuertas = 0;
        this.capacidadMaletero = 0;
    }
}

```

Error:
No se puede obtener acceso al miembro 'Construcciones.Program.Coche.capacidadMaletero' con una referencia de instancia;

C# no permite asignar en el constructor de la clase un atributo static.

Paso 15. Declaramos sólo una variable estática y dejamos la otra normal en el constructor para ser inicializada (para que el programa compile):

```

class Coche : Vehiculo
{
    public int numPuertas;
    public static int capacidadMaletero;
    public Coche(string colo, int cab, string mar, string mod,
        int npuertas) : base(colo, cab, mar, mod) {
        this.numPuertas = npuertas;
    }
    public Coche(): base() {
        this.numPuertas = 0;
    }
}

```

Paso 16. A continuación vamos a la clase Main y creamos 3 variables de tipo coche con el constructor por defecto. Asignamos a sus atributos estáticos valores distintos (los atributos estáticos se deben de inicializar aparte, porque el constructor no lo permite. Finalmente mostramos el contenido de la variable estática de los 3 objetos coche (con Console.Write) y explica qué sucede.

```

static void Main(string[] args)
{
    Vehiculo v = new Vehiculo("amarillo",180, "Nissan","Primera");
    Coche co1 = new Coche();
    co1.capacidadMaletero = 6;
    Coche co2 = new Coche();
    co2.capacidadMaletero = 7;
    Coche co3 = new Coche();
    co3.capacidadMaletero = 8;
}

```

Error:
No se puede obtener acceso al miembro 'Construcciones.Program.Coche.capacidadMaletero' con una referencia de instancia;

En Java se puede hacer este enunciado, pero C# no te deja instanciar una variable estática desde un objeto determinado, pero si naturalmente desde la clase.

Tanto en Java como en C# el significado de static es el mismo: las variables estaticas de una clase son compartidas por todos los objetos de esa misma clase. El valor de la variable estatica es el mismo e igual a la última asignación.

La única diferencia es que para reforzar este efecto en C# no permite referirse a los atributos estáticos desde un objeto determinado (no compila), se debe de hacer desde la propia clase obligatoriamente.

Paso 17. En la clase Coche crea un método estático, por ejemplo de nombre arrancar. Dicho método debe de operar tanto con las variables estáticas de la clase como con las normales. ¿Qué ocurre?

```
class Coche : Vehiculo
{
    public int numPuertas;
    public static int capacidadMaletero;
    public Coche(string colo, int cab, string mar, string mod,
        int npuertas) : base(colo, cab, mar, mod) {
        this.numPuertas = npuertas;
    }
    public Coche(): base() {
        this.numPuertas = 0;
    }
    public static void arrancar()
    {
        if (numPuertas==4) capacidadMaletero=9;
        else
    }
```

Se requiere una referencia de objeto para el campo, método o propiedad no estáticos

En una función estática de C#, sólo se puede llamar a atributos estáticos de la clase.

Paso 18. Corrige el inconveniente anterior, de modo que la función estática sólo opere atributos estáticos, y llámala desde el programa principal para ver su comportamiento.

```
class Coche : Vehiculo
{
    public int numPuertas;
    public static int capacidadMaletero;
    public Coche(string colo, int cab, string mar, string mod,
        int npuertas) : base(colo, cab, mar, mod) {
        this.numPuertas = npuertas;
    }
    public Coche(): base() {
        this.numPuertas = 0;
    }
    public static void arrancar()
    {
        if (capacidadMaletero==4) capacidadMaletero=9;
        else capacidadMaletero=5;
    }
}
```

```
static void Main(string[] args)
{
    Vehiculo v = new Vehiculo("amarillo",180, "Nissan","Primera");
    Coche col = new Coche();
    Coche.arrancar();
    col.arrancar();
}

void Coche.arrancar()

Error:
No se puede obtener acceso al miembro 'Construcciones.Program.Coche.arrancar()' con una referencia de instancia;
```

Como esperábamos C# sólo permite llamar a una función estática desde una referencia de instancia.

EJERCICIO 2

Paso 1. Crea una clase dentro de Coche que se llamará Radio. Esta clase debe tener los métodos encender(), apagar() y sintonizar().

```
class Coche : Vehiculo
{
    public int numPuertas;
    public int capacidadMaletero;
    public Radio r;
    public Coche(string colo, int cab, string mar, string mod,
        int npuertas, int maletero) : base(colo, cab, mar, mod)
    {
        this.numPuertas = npuertas;
        capacidadMaletero = maletero;
        r = new Radio();
    }
    public Coche(): base() {
        this.numPuertas = 0;
        capacidadMaletero = 0;
        r = new Radio();
    }
    public class Radio {
        Boolean encendido;
        int frecuencia = 100;
        public void encender()
        {
            if (!encendido) { encendido = true; }
        }
        public void apagar()
        {
            if (encendido) { encendido = false; }
        }
        public void sintonizar(int f)
        {
            frecuencia = f;
        }
    }
}
```

Paso 2. Usa esta clase desde el programa principal. Acceden a sus atributos públicos directamente y a algún método de radio.

```
static void Main(string[] args)
{
    Vehiculo v = new Vehiculo("amarillo",180, "Nissan","Primera");
    Coche col = new Coche();
    col.numPuertas = 6;
    col.capacidadMaletero = 67;
    col.r.sintonizar(110);
}
```

Paso 3. Crea una interfaz llamada Producto que defina los siguientes métodos:

- getPrecio()
- getDescripcion()

```
public interface IProducto
{
    int getPrecio();
    string getDescripcion();
}
```

En C# por convención las interfaces llevan el prefijo I → IProducto

Paso 4. Haz que la clase coche implemente la interfaz Producto. ¿Qué ocurre si no implementas los métodos de la interface?

```
class Coche : Vehiculo, IProducto
{
    public int capacidadMaletero;
    public Radio r;
    public Coche(string colo, int cab, string mar, string mod,
        int npuertas, int maletero) : base(colo, cab, mar, mod)
    {
        this.numPuertas = npuertas;
        capacidadMaletero = maletero;
        r = new Radio();
    }
}
```

'Construcciones.Program.Coche' no implementa el miembro de interfaz 'Construcciones.Program.IProducto.getPrecio()'

En C#, como en Java, no se permite la herencia múltiple, sólo se puede heredar de una clase base, pero en cambio puede implementar muchas Interfaces. Primero debe ir la clase base y después el resto de interfaces

```
class Coche : Vehiculo, IProducto
{
    public int numPuertas;
    public int capacidadMaletero;
    public Radio r;
    public Coche(string colo, int cab, string mar, string mod,
        int npuertas, int maletero) : base(colo, cab, mar, mod)
    {
        this.numPuertas = npuertas;
        capacidadMaletero = maletero;
        r = new Radio();
    }
    public Coche(): base() {
        this.numPuertas = 0;
        capacidadMaletero = 0;
        r = new Radio();
    }
    public int getPrecio()
    {
        return 5000;
    }
    public string getDescripcion()
    {
        return "El coche fantastico";
    }
}
```

```
static void Main(string[] args)
{
    Vehiculo v = new Vehiculo("amarillo",180, "Nissan","Primera");
    Coche col = new Coche();
    col.numPuertas = 6;
    col.capacidadMaletero = 67;
    col.r.sintonizar(110);
    Console.WriteLine(col.getPrecio());
    Console.WriteLine(col.getDescripcion());
    Console.ReadKey();
}
```

Paso 5. Crea otra clase Moto que heredará de vehículo. Debe contener las variables xx e yy. Además, haz que implemente también la interfaz Producto.

```
class Moto : Vehiculo, IProducto
{
    public int xx;
    public int yy;
    public Moto()
        : base()
    {
        xx = 0;
        yy = 0;
    }
    public int getPrecio() { return 5000; }
    public string getDescripcion() {
        return "La Moto fantastica";
    }
}
```

Paso 6. En un programa crea 1 objeto tipo Coche y otro Moto, y guárdalos en variables tipo Vehículo. ¿Qué ventajas tiene hacer algo así? ¿Y desventajas? Accede ahora a los métodos `getDescripcion` de estas variables, pero teniendo en cuenta que la clase base `Vehiculo` no tiene definidos estos métodos:

```
static void Main(string[] args)
{
    Coche coche = new Coche();
    Moto moto = new Moto();
    Vehiculo A = coche;
    Vehiculo B = moto;
    Console.WriteLine(A.getDescripcion());
    Console.WriteLine(B.getDescripcion());
    Console.ReadKey();
}
```

Esto falla porque la clase base no tiene definido el método `getDescripcion`. Debemos hacer el cast específico sabiendo de qué tipo se trata.

```
static void Main(string[] args)
{
    Coche coche = new Coche();
    Moto moto = new Moto();
    Vehiculo A = coche;
    Vehiculo B = moto;
    Console.WriteLine(((Coche)A).getDescripcion());
    Console.WriteLine(((Moto)B).getDescripcion());
    Console.ReadKey();
}
```

Las ventajas es poder combinar en un objeto de tipo la superclase, objetos de las diferentes subclases heredadas.

Paso 7. Haz lo mismo del ejercicio anterior, esta vez guardando 1 coche y 1 moto en variables tipo producto. Explica también las ventajas y desventajas de esta acción.

```
static void Main(string[] args)
{
    Coche coche = new Coche();
    Moto moto = new Moto();
    IProducto A = coche;
    IProducto B = moto;
    Console.WriteLine(A.getDescripcion());
    Console.WriteLine(B.getDescripcion());
    Console.ReadKey();
}
```

Ahora el método `getDescripcion` si que lo tiene definido la interfaz `IProducto`.

Paso 8. Sobreescribe la función ToString() en la clase Moto y Coche.

```
class Moto : Vehiculo, IProducto
{
    public int xx;
    public int yy;
    public Moto()
        : base()
    {
        xx = 0;
        yy = 0;
    }
    public int getPrecio() { return 5000; }
    public string getDescripcion() {
        return "La Moto fantastica";
    }
    public override string ToString(){
        return "se trata de una moto";
    }
}
```

```
class Coche : Vehiculo, IProducto
{
    public int numPuertas;
    public int capacidadMaletero;
    public Radio r;
    public Coche(string colo, int cab, string mar, string mod,
        int npuertas, int maletero) : base(colo, cab, mar, mod)
    {
        this.numPuertas = npuertas;
        capacidadMaletero = maletero;
        r = new Radio();
    }
    public Coche(): base() {
        this.numPuertas = 0;
        capacidadMaletero = 0;
        r = new Radio();
    }
    public override string ToString()
    {
        return "se trata de un coche";
    }
}
```

ToString es un método sobrecargado del objeto Object, del cual derivan todos los objetos.

Paso 9. Haz un programa que genere un array de 5 Vehículos. El usuario podrá escoger si desea un Coche o una Moto, y deberá rellenar los datos necesarios para crear el objeto. Al final, muestra por pantalla el contenido del array utilizando la función ToString, y discriminando entre los dos tipos de objetos:

```

static void Main(string[] args)
{
    Vehiculo []vh = new Vehiculo[5];
    int opcion;
    for(int i=0; i< 5;i++)
    {
        Console.Write("1.Coche o 2.Moto?");
        opcion = int.Parse(Console.ReadLine());
        if (opcion == 1)
            vh[i] = new Coche();
        else
            vh[i] = new Moto();
    }
    for (int i = 0; i < 5; i++)
    {
        if (vh[i] is Moto)
            Console.WriteLine("Objeto " + (i+1) + " " + ((Moto)vh[i]).ToString());
        else
            Console.WriteLine("Objeto " + (i+1) + " " + ((Coche)vh[i]).ToString());
    }
    Console.ReadKey();
}

```

```

file:///C:/Users/Antonio/Documents/Visual S
1.Coche o 2.Moto?1
1.Coche o 2.Moto?1
1.Coche o 2.Moto?2
1.Coche o 2.Moto?2
1.Coche o 2.Moto?1
Objeto 1 se trata de un coche
Objeto 2 se trata de un coche
Objeto 3 se trata de una moto
Objeto 4 se trata de una moto
Objeto 5 se trata de un coche

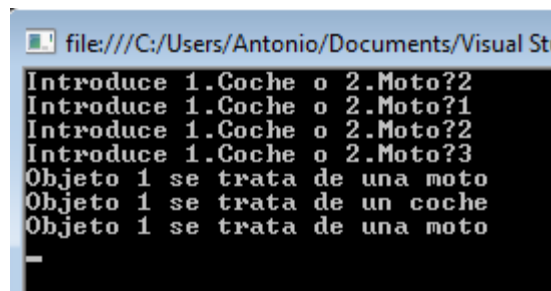
```

Paso 10. Realiza el paso anterior de nuevo, pero esta vez usando la clase Vector en lugar de un array, y dejando escoger al usuario el número de vehículos que se van a crear.

En C# la clase Vector se corresponde con la de List en C#

- `std::vector` - `List<T>`
- `std::list` - `LinkedList<T>`
- `std::map` - `SortedDictionary<Tkey, Tvalue>`
- `std::set` - `HashSet<T>` (as pointed out by *Weyland Yutani* in comments)
- `std::multiset` - `Dictionary<Tkey, int>` (`int` keeping count of the number of `Tkey` s)
- `std::multimap` - `Dictionary<Tkey, List<Tvalue>>`

```
static void Main(string[] args)
{
    List<Vehiculo> vh = new List<Vehiculo>();
    int opcion;
    Boolean final = false;
    while (!final)
    {
        Console.Write("Introduce 1.Coche o 2.Moto?");
        opcion = int.Parse(Console.ReadLine());
        if (opcion == 1)
            vh.Add(new Coche());
        else if (opcion == 2)
            vh.Add(new Moto());
        else final = true;
    }
    int i = 1;
    foreach(Vehiculo v in vh)
    {
        if (v is Moto)
            Console.WriteLine("Objeto " + i + " " + ((Moto)v).ToString());
        else
            Console.WriteLine("Objeto " + i + " " + ((Coche)v).ToString());
    }
    Console.ReadKey();
}
```



```
file:///C:/Users/Antonio/Documents/Visual St
Introduce 1.Coche o 2.Moto?2
Introduce 1.Coche o 2.Moto?1
Introduce 1.Coche o 2.Moto?2
Introduce 1.Coche o 2.Moto?3
Objeto 1 se trata de una moto
Objeto 1 se trata de un coche
Objeto 1 se trata de una moto
_
```