Curso de Patrones y Buenas Prácticas .Net

Programación Orientada a Objetos

Plataforma: Visual Studio .Net
Framework: Microsoft .Net Framework

Documento de Referencia y Capacitación
(Este documento está sujeto a cambios)

Fecha de Creación: 28-Mar-2013
Versión: 2.0.0.0
Autor: Julio Cesar Robles Uribe

Tabla de Contenido

Introducción

En este curso se muestran los patrones y buenas prácticas de programación, que son utilizadas en la industria del software, en particular en plataforma .Net y su aplicabilidad en el desarrollo de productos para el mundo.

Inicialmente se da una breve introducción al concepto de programación orientada a objetos partiendo de los puntos claves de esta metodología, esto soportado sobre el ambiente de desarrollo (IDE) de Microsoft Visual Studio .Net, además de muestra el patrón de arquitectura de capas (Layers) que permite estructurar de manera distribuida y desacoplada cualquier tipo de aplicación, para terminar se examinan los patrones de diseño más conocidos en la industria del software y se detallan los errores más frecuentes cometidos por los desarrolladores contrarrestados con una buena práctica.

De igual forma se introduce de manera sencilla, las principales características del Framework de .Net en su versión 4.5.

Objetivos

- Introducir los conceptos de programación orientada a objetos (OOP - Object Oriented Programming)
- Mostrar el ambiente de integrado de desarrollo (IDE – Integrated Development Environment) de Microsoft Visual Studio .Net, como plataforma de desarrollo.
- Explicar el patrón de arquitectura de capas y su aplicación práctica.
- Mostrar los patrones de diseño más utilizados en la industria de software.
- Mostrar la corrección de los errores más frecuentes en programación mediante el uso buenas prácticas.
- Mostrar las características principales de .Net Framework 4.5.

Prerrequisitos

Conocimientos

- Conocimiento básico en programación.
  - Manejo de variables (int, long, string, char, boolean, etc.).
  - Manejo de sentencias condicionales (if-else, switch).
  - Manejo de ciclos (for, while, foreach).
  - Manejo de procedimientos y funciones (Programación Estructurada).
  - Paso de parámetros (por valor o por referencia).
- Conocimiento básico de sentencias SQL.
  - DDL (Data Definition Language).
    - Create, Drop, Alter.
  - DML (Data Manipulation Language).
    - Select, Insert, Delete, Update.

- **Opcional**
  - C++ o Java.
  - Estándar SQL/92.

Herramientas

- Microsoft Visual Studio .Net 2015 o C Sharp Express 2012
- Microsoft .Net Framework 4.5.
- Base de datos SQL Server 2012 Express
- Herramienta de Administración SQL Server Management Studio 2012.
- Base de Datos Oracle Express 11 g R2.
- Herramienta para conexión a base de datos ORACLE (SQL Developer).
- Oracle Data Access Component (ODAC)
- Oracle Instant Client Lite 11 g.

Sesión I: Programación Orientada a Objetos

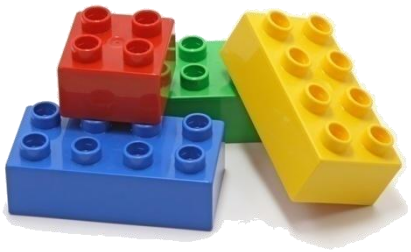
Conceptos

En esta sesión vamos a hablar acerca de la Programación Orientada a Objetos (OOP) y de los fundamentos del lenguaje C#.

La programación orientada a objetos es un conjunto de características o también una filosofía de programación que busca minimizar el proceso de creación de aplicaciones complejas por medio de relación de cosas en código que tienen un sentido en el mundo real.

Ya que la OOP no es un lenguaje de programación, puede aplicarse a cualquier lenguaje, y de hecho hoy en día está disponible en mayor o menor medida en todos los lenguajes tradicionales (C se ha convertido en C++, Pascal en Delphi, VB incorpora parte de la OOP) y no aparece un lenguaje nuevo sin que incluya OOP (como es el caso de Java y C#). Es por esto que intentaremos que todo lo que aquí se diga pueda ser aplicado a cualquier lenguaje OOP. Sin embargo, cuando llegue el momento de escribir código, emplearemos la sintaxis de C# que puede resultar más clara independientemente del lenguaje al que se acaben aplicado estos conocimientos.

¿Qué es OOP?



Para brindar claridad en este aspecto vamos a decir primero qué no es OOP, para, luego, intentar dar una definición de lo que sí es.

La OOP no es: Un sistema de comunicación con los programas basados en ratones, ventanas, iconos, etc. Puesto que, normalmente, los lenguajes de OOP suelen presentar estas características y puesto que habitualmente estos entornos suelen desarrollarse con técnicas de OOP, algunas personas tienden a identificar OOP y entornos de este tipo.

La OOP no es: Un lenguaje de programación. De hecho las técnicas de OOP pueden utilizarse en cualquier lenguaje conocido y los que están por venir, aunque estos últimos, al menos en los próximos años, incluirán facilidades para el manejo de objetos. Desde luego, que en los lenguajes que prevén el uso de objetos, la implementación de las técnicas de OOP resulta mucho más fácil y provechosa que los otros. Pero del mismo modo a lo comentado en el punto anterior, se pueden utilizar estos lenguajes sin que los programas resultantes tengan nada que ver con la OOP.

Que si es OOP.

La OOP son un conjunto de técnicas que nos permiten incrementar enormemente nuestro proceso de producción de software; aumentando drásticamente nuestra productividad por un lado y permitiéndonos abordar proyectos de mucha mayor envergadura por el otro. Usando estas técnicas, nos aseguramos la re-usabilidad de nuestro código, es decir, los objetos que hoy escribimos, si están bien escritos, nos servirán para "siempre". Hasta aquí, no hay ninguna diferencia con las funciones, una vez escritas, estas nos sirven siempre.

Lenguaje de programación C#

El lenguaje de programación **C#** fue desarrollado por Microsoft específicamente para la plataforma .Net como un lenguaje que permitiera a los programadores migrar con facilidad hacia .Net. Tiene sus raíces en Java, C y C++; adapta las mejores características de cada uno de estos lenguajes y agrega características propias. C# está orientado a objetos y contiene una poderosa biblioteca de clases (FCL por sus siglas en Ingles; FrameWork Class Library), mejor conocida como Biblioteca de Clases de Framework, que consta de componentes pre-construidos que permiten a los programadores desarrollar aplicaciones con rapidez, además este lenguaje es apropiado para desarrollar aplicaciones de escritorio (Windows Forms), así como Smart Clients, Aplicaciones Web (ASP .Net) y Aplicaciones Móviles, entre otras.

C# es un lenguaje de programación visual controlado por eventos, en el cual se crean programas mediante el uso de un Entorno de Desarrollo Integrado (IDE Por sus siglas en Inglés; Integrated Development Environment). Con un IDE un programador puede crear, ejecutar, probar y depurar programas en C#, con lo cual se reduce el tiempo requerido para producir un programa funcional en una fracción del tiempo que llevaría sin utilizar el IDE. La plataforma .Net permite la interoperabilidad de los lenguajes: los componentes de software de distintos lenguajes pueden interactuar como nunca antes se había hecho. Los desarrolladores pueden empaquetar incluso hasta el software antiguo para que trabaje con nuevos programas en C#. Además, las aplicaciones en C# pueden interactuar a través de Internet mediante el uso de estándares industriales de comunicación como XML (eXtensible Markup Language) o el SOAP (Simple Object Access Protocol).

El lenguaje de programación C# original se estandarizó a través de la Ecma International (www.ecma-international.org) en Diciembre del 2002 como Estándar ECMA-334: Especificación del Lenguaje C# (Ubicado en www.ecma-international.org/publications/standards/Ecma-334.htm). Desde entonces, Microsoft propuso varias extensiones del lenguaje que se han adoptado como parte del estándar Ecma C# revisado. Microsoft hace referencia al lenguaje C# completo (incluyendo las extensiones adoptadas) como **C# 2.0**.

Retroalimentación

1. La OOP es un sistema de comunicación con los programas basados en ratones, ventanas, iconos, etc.

   ◉ Verdadero ○ Falso

   R/ Falso: Los lenguajes de OOP suelen presentar estas características y puesto que habitualmente estos entornos suelen desarrollarse con técnicas de OOP, algunas personas tiende a identificar OOP y entornos de este tipo.

2. Con OOP podemos incorporar objetos que otros programadores han construido en nuestros programas.

   ○ Verdadero ○ Falso

   R/ Verdadero: De igual modo como vamos a una ferretería y compramos piezas de madera para ensamblarlas y montar una estantería o una mesa.

3. C# es un lenguaje de programación diseñado para crear una amplia gama de aplicaciones que se ejecutan en .NET Framework. C# es simple, eficaz, con seguridad de tipos y orientado a objetos con sus diversas innovaciones.

   ○ Verdadero ○ Falso

   R/ Verdadero: C# permite desarrollar aplicaciones rápidamente y mantiene la expresividad y elegancia de los lenguajes de tipo C.

Programación Orientada a Objetos

En esta sesión se estudiarán los conceptos principales de la programación orientada a objetos como son: Clase, Objeto, Herencia, Encapsulación y Polimorfismo. Estas son las ideas más básicas que todo aquel que trabaja en OOP debe comprender y manejar constantemente; es por lo tanto de suma importancia que los entienda claramente. De todos modos, no se preocupe si al finalizar el presente capítulo, no tiene una idea clara de todos o algunos de ellos: con el tiempo los irá asentando y se irá familiarizando con ellos, especialmente cuando comience a trabajar creando sus propias clases y jerarquías de clases.

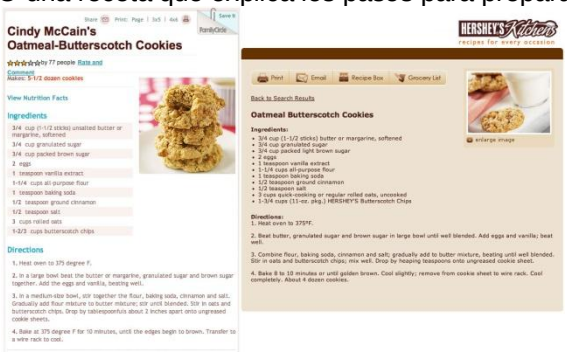
Clases

“El bloque básico de Programación Orientada a Objetos es una Clase “

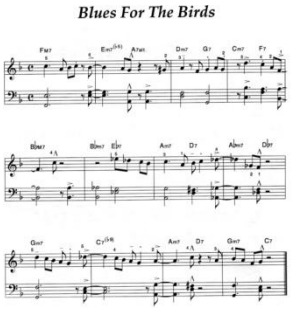
Una clase, es simplemente una abstracción que hacemos de nuestra experiencia sensible. El ser humano tiende a agrupar seres o cosas -objetos- con características similares en grupos -clases-. Así, aun cuando existen por ejemplo multitud de vasos diferentes, podemos reconocer un vaso en cuanto lo vemos, incluso aun cuando ese modelo concreto de vaso no lo hayamos visto nunca. El concepto de vaso es una abstracción de nuestra experiencia sensible.

Pensemos en una clase como un plano para construir una casa

O una receta que explica los pasos para preparar unas galletas



O como la partitura de una canción que una persona puede tocar e incluso, muchos grupos pueden tocar la misma pieza de música.



Sin embargo en cada uno de estos casos hay una distinción que debemos hacer: un plano no es una casa, la receta no es una galleta de chocolate y una partitura no es una pieza de

música. Estas definen como las anteriores deben ser realizadas o ejecutadas, pero no es la misma cosa.

Se pueden crear muchas casas del mismo plano y cada casa es una Instancia del plano. Los colores de los ladrillos, las paredes o de la alfombra pueden ser diferentes, pero al final del caso se trata de la misma casa. Se pueden crear muchas galletas de la misma receta, cada galleta puede ser diferente en cuanto a forma, unas más pequeñas que otras; pero no se puede negar que todas las galletas fueron cocinadas con la misma receta. Muchos músicos diferentes pueden tocar la misma partitura, puede que suene un poco diferente, pero de igual forma corresponde a la misma canción.

Esto es básicamente la diferencia entre una clase y un objeto. Una clase es un plano y un objeto es una instancia de un plano. Se pueden crear muchas instancias de una clase y cada instancia es creada bajo los lineamientos de la clase. Cada objeto de la instancia puede tener muchas propiedades o diferentes valores para las propiedades pero siguen siendo de la misma clase.

En conclusión las clases definen las características y comportamiento de los objetos, pero no son los objetos en sí, pero todos los objetos definidos por medio de una clase poseen las características y comportamientos definidos por esta.

Objetos

- El objeto es la instancia real de la Clase, en la cual toman importancia las características de la clase.

- Un objeto no es un dato simple, sino que puede contener en su interior cierto número de atributos bien estructurados.

- Cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica de otro tipo.

Las partes principales del objeto que son definidas por la clase son:

1. Los Campos o Atributos.
2. Las Propiedades.
3. Los Métodos.
4. El Constructor.

Campos o Atributos

Los campos o atributos son las partes más internas de un objeto y no son expuestas a los demás por eso siempre se declaran privadas a la clase.

Son quienes definen las características principales del objeto y las que le dan la estructura principal al mismo.

Los campos pueden ser expuestos a los demás mediante el uso de propiedades, pero no necesariamente todo campo está relacionado con una propiedad y no toda propiedad está relacionada con un solo campo.

Los campos o atributos se definen de la siguiente forma:

```
prívate  <tipo> <nombre atributo>;
```

Ejemplo:

```
//Nombre de la Persona
prívate string firstName;
```

Propiedades

Las propiedades distinguen a un objeto determinado de los demás y forman parte de su misma instancia.

Las propiedades encapsulan a los campos o atributos y pueden o no tener lógica que condicione el contenido o cálculo de los mismos.

Las propiedades de un objeto pueden ser heredadas a sus descendientes, a diferencia de los campos o atributos que no pueden serlo, en consecuencia de esto un objeto puede tener **propiedades propias**, que son definidas o asociadas directamente al objeto actual o **propiedades heredadas** que son definidas en su clase padre o antecesora.

Las propiedades pueden ser de lectura y/o escritura, para lo cual se maneja la siguiente definición:

```
/// <summary>
/// Objetivo de la Propiedad
/// </summary>
public <Propiedad>
{
   get
      {
         Return <nombre del campo o atributo>;
      }
   set
      {
         <nombre del campo o atributo> = value;
      }
}
```

Ejemplo:

```
/// <summary>
/// Nombre completo de la persona
/// </summary>
public FirtsName
{
   get
      {
         Return firstName;
      }
   set
      {
         firstName = value;
      }
}
```

Si se desea que la propiedad sea de solo lectura, se elimina la parte del **set**, así los cálculos sobre los campos referenciados, solo pueden ser obtenidos mediante la función del **get**.

Métodos

Los métodos son las operaciones que pueden realizarse sobre el objeto, y que normalmente están definidos dentro de la clase, o en una interfaz común a varias clases.

Los métodos definen el comportamiento del objeto y también pueden ser heredados o no a las clases predecesoras, por lo tanto existen **Métodos Propios** definidos dentro del objeto o **Métodos Heredados** definidos en la clase padre, a estos también se les llaman métodos miembro, porque el objeto los posee por el simple hecho ser miembro de una clase.

Para crear un método se utiliza la siguiente definición:

```
public <Tipo> <Nombre del Método>(parámetros)
{
   //Contenido del Método
}
```

Ejemplo:

```
/// <summary>
/// Realiza el cálculo de la suma de los dos valores
/// </summary>
/// <param name="valor1">Valor Inicial</param>
/// <param name="valor2">Valor a ser adicionado</param>
/// <returns>Resultado de la Suma</returns>
public decimal CalcularSuma(decimal valor1, decimal valor2)
{
   //Variable para obtener el resultado
   decimal resultado=0;

   //Realizar la operación
   resultado = valor1 + valor2;

   //retornar el resultado
   return resultado;
}
```

Constructor

El constructor es un método particular que permite crear una nueva instancia del objeto, reservándole espacio en memoria al mismo, según la definición hecha en la clase. Adicionalmente el objetivo principal del constructor es inicializar los campos o atributos propios de la clase, con los valores respectivos o un valor por defecto.

Por ejemplo si la clase es Persona, el constructor sería el siguiente:

```
/// <summary>
/// Constructor por defecto
/// </summary>
public Person()
{
   //Identificador de la persona
   person_Id = 0;
   //Nombres de la Persona
   firstName = string.Empty;
   //Apellidos de la Persona
   lastName = string.Empty;
   //Fecha de Nacimiento (dd-mm-yyyy)
   birthDay = DateTime.Now;
   //Sexo (M,F)
   sex = 'M';
}
```

Ejercicio Práctico

Tomemos como base la definición de la clase PERSONA, en este caso solo vamos a identificar algunas de sus características generales.

Una Persona tiene las siguientes propiedades o características:

- **Nombres:** Hace referencia al primer y segundo nombre
- **Apellidos:** hace referencia al primer y segundo apellido.
- **Fecha de Nacimiento:** Fecha en la que nació la persona.
- **Sexo:** referente a la identificación como Masculino o Femenino de la persona.

Para identificar fácilmente una persona de otra se le puede crear una propiedad automática denominada ID, que permita diferenciar una persona de otra así:

- **Id:** Código de identificación para diferenciar de manera única cada persona.

Llevando esto a programación en C# utilice un editor de texto, como por ejemplo el block de notas (NotePad.exe) y cree el siguiente documento llamado **Person.cs**.

Pasos:
Cree la definición de la clase según el estándar y utilice la plantilla de creación para C# (**Plantilla.cs**)

1. Cree la clase en notación Pascal Case así:

```
public class Person
```

Recuerde : La notación para la definición de las clases utiliza la primera letra de El modificador `public` define que la clase puede ser vista y utilizada desde cualquier parte, los modificadores principales son:

- **Public**: Publico y pueden ser vistos por todos.
- **Private**: Solo pueden ser vistos por la clase misma
- **Protected**: Solo pueden ser vistos por la misma clase o sus hijos o derivados, es decir los que heredan de esta clase.
- **Internal**: Solo pueden ser vistos en el mismo proyecto.

2. Cree los atributos propios de la clase así:

```
//Identificador de la persona
private long person_Id;
//Nombres de la Persona
private string firstName;
//Apellidos de la Persona
private string lastName;
//Fecha de Nacimiento (dd-mm-yyyy)
private DateTime birthDay;
//Sexo (M,F)
private char sex;
```

Recuerde: La notación para los campos o atributos de la clase se definen en camel Case (primera letra de la primer palabra en minúscula y luego las siguientes primeras letras de las otras palabras en mayúscula)

3. Cree las propiedades basándose en los campos o atributos de la clase así: (ejemplo para el campo **person_Id**, debe realizar lo mismo para los demás miembros de la clase).

```
public long Person_Id
{
   get
   {
      return person_Id;
   }
   set
   {
      person_Id = value;
   }
}
```

Recuerde: es considerado buena práctica ser explicito en el detalle de los métodos `get` y `set`, ya que la creación automática de las propiedades no desagregan en varias líneas, lo que no permite establecer con claridad el tamaño del método, adicionalmente es importante tener en cuenta que estos métodos, además de encapsular las propiedades (protegerlas del entorno exterior) pueden tener varias líneas o lógica de ejecución, tales como operaciones o cálculos internos. Si desea tener una propiedad de solo lectura debe eliminar el método `set` y si desea tenerla de solo escritura debe eliminar el método `get`.

4. Cree el constructor de la clase, recuerde que función principal de este es únicamente inicializar los campos o atributos de la clase.

```
public Person()
{
    //Identificador de la persona
    person_Id = 0;
    //Nombres de la Persona
    firstName = string.Empty;
    //Apellidos de la Persona
    lastName = string.Empty;
    //Fecha de Nacimiento (dd-mm-yyyy)
    birthDay = DateTime.Now;
    //Sexo (M,F)
    sex = 'M';
}
```

Recuerde: El constructor debe tener el mismo nombre de la clase finalizando con los paréntesis ().

5. Cree un constructor especializado con todos los campos de la clase así:

```
public Person(string firstName, string lastName, DateTime birthDay, char sex)
{
    //Nombres de la Persona
    this.firstName = firstName;
    //Apellidos de la Persona
    this.lastName = lastName;
    //Fecha de Nacimiento (dd-mm-yyyy)
    this.birthDay = birthDay;
    //Sexo (M,F)
    this.sex = sex;
}
```

Recuerde: para utilizar la asignación de los parámetros a los miembros de la clase debe utilizar el prefijo `this`, que hace referencia a los atributos de la clase y permite realizar la asignación o manipulación de los mismos.

Importante: Cuando desarrolle el ejercicio no olvide documentar la clase, para ello utilice la plantilla de código (**Plantilla.cs**) y tenga en cuenta llenar los campos detallados en ella.

Encapsulamiento

Esta característica de la programación orientada a objetos, es la que permite proteger los campos o atributos internos de un objeto, y solamente exponer los valores mediante propiedades o métodos públicos que alteran su contenido.

La idea principal del encapsulamiento es garantizar la independencia entre la forma de hacer las cosas y los servicios que se ofrecen al exterior, de esta forma se garantiza el intercambio de componentes sin afectar el comportamiento de un sistema.

Un ejemplo valido del encapsulamiento son las propiedades

```
/// <summary>
/// Nombre completo de la persona
/// </summary>
public FirtsName
{
   get
      {
         Return firstName;
      }
   set
      {
         firstName = value;
      }
}
```

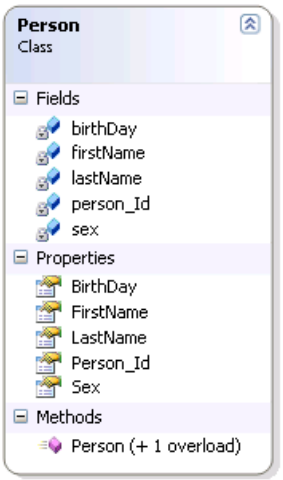
En esta se expone la propiedad **FirstName** pero el campo o atributo **firstName** interno no puede ser manipulado y ni visto desde afuera.

Herencia

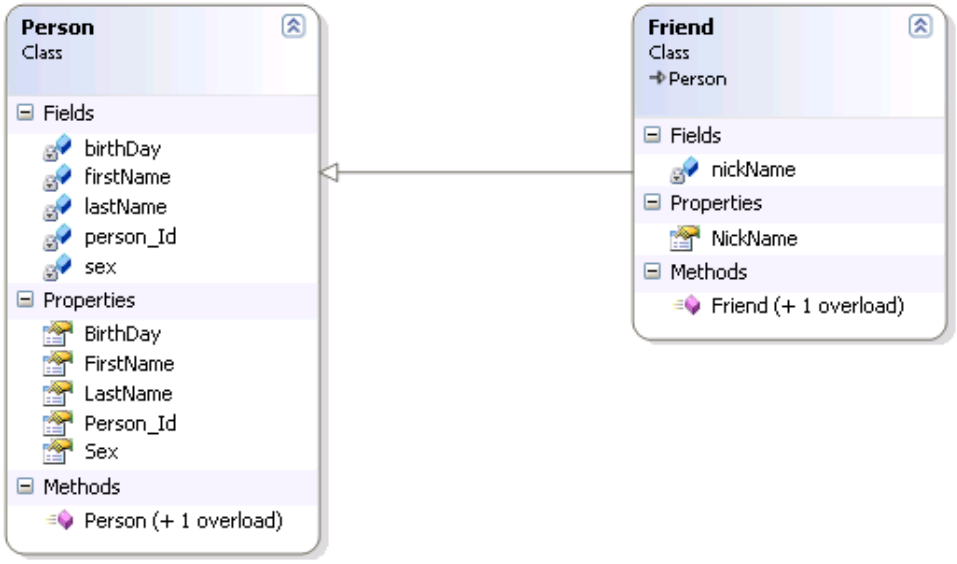
Esta es tal vez una de las características principales de la programación orientada a objetos, en donde un objeto comparte la definición de otro y especializa su comportamiento.

También podemos decir que la herencia es el mecanismo que permite que un objeto de clase **A** herede las características de otro objeto de clase **B**, con esto los objetos de clase A, pueden acceder a las propiedades y métodos de la case B, sin tener que redefinirlos.

Por ejemplo si tengo una clase Persona con la siguiente definición



Y se necesita crear una clase amigo (Friend) que herede de persona así:



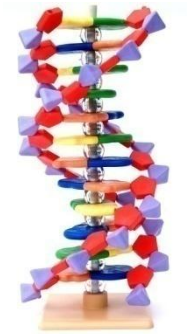
La definición de la clase amigo sería la siguiente:

```
/// <summary>
/// Entidad Friend
/// </summary>
public class Friend:Person
{
   //Campos o Atributos
   #region Campos o Atributos
   //Apodo o sobrenombre
   private string nickName;
   #endregion Campos o Atributos

   //Propiedades
   #region Propiedades
   /// <summary>
   /// Apodo o Sobrenombre del amigo
   /// </summary>
   public string NickName
   {
      get
      {
         return nickName;
      }
      set
      {
         nickName = value;
      }
   }
   #endregion Propiedades

   //Constructores
   #region Constructores
   /// <summary>
   /// Constructor por defecto
   /// </summary>
   public Friend()
   {
      //Inicializar los valores de la base (Person)

      //Identificador de la persona
      base.person_Id = 0;
      //Nombres de la Persona
      base.firstName = string.Empty;
      //Apellidos de la Persona
      base.lastName = string.Empty;
      //Fecha de Nacimiento (dd-mm-yyyy)
      base.birthDay = DateTime.Now;
      //Sexo (M,F)
      base.sex = 'M';

      //Inicializar los atributos de la clase propia

      //Iniciar el apodo
      this.nickName= string.Empty;
   }
```

```
    /// <summary>
    /// Constructor con campos
    /// </summary>
    /// <param name="firstName">Nombres</param>
    /// <param name="lastName">Apellidos</param>
    /// <param name="birthDay">Fecha de Nacimiento</param>
    /// <param name="sex">Sexo (M,F)</param>
    /// <param name="nickName">Apodo o Sobrenombre</param>
    public Friend(string firstName, string lastName, DateTime birthDay, char sex,
                  string nickName)
    {
        //Asignar los valores de la clase base

        //Nombres de la Persona
        base.firstName = firstName;
        //Apellidos de la Persona
        base.lastName = lastName;
        //Fecha de Nacimiento (dd-mm-yyyy)
        base.birthDay = birthDay;
        //Sexo (M,F)
        base.sex = sex;

        //Asignar los valores para la clase propia (Friend)

        //Asignar el apodo
        this.nickName = nickName;
    }
    #endregion Constructores

}//Fin: Friend
```

Tenga presente que cuando esta asignando los valores a los atributos de la clase padre, debe utilizar la palabra reservada **base** y cuando va a hacer referencia a los campos propios de la clase utiliza la palabra reservada **this**.

Polimorfismo



El polimorfismo no es otra cosa que la posibilidad de construir el varios métodos con el mismo nombre, pero con relación a la clase a la que pertenece cada uno, con comportamientos diferentes. Esto conlleva la habilidad de enviar un mismo mensaje a objetos de clases diferentes y que cada uno responda de forma distinta.

Ejemplo:
Supongamos que tenemos la clase Animal que tiene el método Hablar, y dos clases hijas Perro y Gato. Cuando en la clase Perro se implementa el método Hablar, este hará “GUAU”, a diferencia de la clase Gato que al ejecutar el método Hablar hará “MIAU”.

Veamos esto en forma de clases:

1. Se define la clase Animal

```
public abstract class Animal
{
   public virtual void Hablar()
   {
      Console.WriteLine("No se que soy");
   }
}
```

Se crea la clase de tipo **`abstract`** ya que no se puede tener una representación real de esta, con esto se facilita también la implementación de las clases concretas de **Perro** y **Gato**.

2. Se define la Clase Perro que hereda de Animal y se sobrescribe el método Hablar

```
public class Perro : Animal
{
   public override void Hablar()
   {
          Console.WriteLine("GUAU");
   }
}
```

Se sobrescribe el método **Hablar** para la clase **Perro**, para ello se utiliza la palabra reservada **`override`** y se implementa su forma particular del método.

3. Se define la clase Gato que herede de Animal y se sobrescribe el método Hablar.

```
public class Gato : Animal
{
 public override void Hablar()
 {
    Console.WriteLine("MIAU");
 }
}
```

Se sobrescribe el método **Hablar** para la clase **Gato**, para ello se utiliza la palabra reservada **override** y se implementa su forma particular del método.

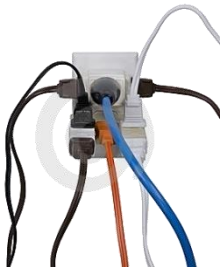
4. Se crea una clase **Zoo** donde se haga el llamado de los objetos de clase Perro y Gato y se ejecutan, en ambos, el método **Hablar**.

```
public class Zoo
{
   public static void main(string[] args)
   {
      Animal minino = new Gato();
      minino.Hablar();

      Animal scooby = new Perro();
      scooby.Hablar ();
   }
}
```

Al ejecutar el programa Zoo, se mostrarían por consola los siguientes valores:

MIAU
GUAU

Sobrecarga



La sobrecarga puede ser considerada como un tipo especial de polimorfismo, también llamada Polimorfismo Paramétrico.

La sobrecarga es la capacidad de definir varias funciones utilizando el mismo nombre, pero usando parámetros diferentes (nombre y/o tipo). La sobrecarga selecciona automáticamente el método correcto a aplicar en función del tipo de datos y la cantidad pasados a la función como parámetros.

Por ejemplo definimos varios métodos llamados **Adicionar**, pero con diferentes tipos de parámetros y diferentes tipos de resultados así:

- El método `int` Adicionar(`int`,`int`) devolvería la suma de dos números enteros.
- `float` Adicionar(`float`, `float`) devolvería la suma de dos flotantes.
- `char` Adicionar(`char`, `char`) daría por resultado la suma de dos caracteres definidos por el autor.

La sobrecarga se da siempre dentro de una sola clase, mientras que el polimorfismo se da entre clases distintas.

Un método está sobrecargado si dentro de una clase existen dos o más declaraciones de dicho método con el mismo nombre pero con parámetros distintos.

La sobrecarga se resuelve en tiempo de compilación (cuando se está construyendo el ejecutable) utilizando los nombres de los métodos y los tipos de sus parámetros; el polimorfismo se resuelve en tiempo de ejecución del programa, esto es, mientras se ejecuta, en función de que clase pertenece un objeto.

Retroalimentación

1. Las clases son la representación abstracta de los objetos del mundo real.

   ◉ Verdadero ○ Falso

   R/Verdadero: las clases definen las características de un objeto pero no son un objeto en sí.

2. Para una objeto los atributos y las propiedades son iguales.

   ◉ Verdadero ○ Falso

   R/Falso: Los atributos son las variables privadas dentro de la objeto y las propiedades son la forma de exponer estas variables a los demás.

3. Las clases se definen en notación:

   ◉ PascalCase ○ CamelCase ○ Hungara

   R/PascalCase: Las clases al igual que los Métodos de una clase se definen con notación PascalCase, en donde la primera letra de cada palabra es en mayúscula.

4. Cada objeto es un dato simple, que no puede contener en su interior cierto número de atributos y tipos de atributos bien estructurados.

   ◉ Verdadero ○ Falso

   R/Falso: un objeto No es un dato simple, ya que puede contener en su interior un conjunto de varios datos bien estructurados o de otro tipo en particular.

5. Los campos o atributos son los que definen el comportamiento de un objeto.

   ◉ Verdadero ○ Falso

   R/Falso: Los campos o atributos definen las características del objeto. Los Métodos son los que definen el comportamiento del objeto.

6. Las propiedades son las que encapsulan los campos o atributos de un objeto.

   ◉ Verdadero ○ Falso

   R/Verdadero: Los campos o atributos no pueden ser expuestos directamente a los demás, para ellos se utilizan las propiedades.

7. Si deseo crear una propiedad de solo lectura debo eliminar el método:

   ◉ get ○ set

   R/set: Si se desea crear una propiedad de solo lectura no se debe permitir asignar valores a ella por lo tanto se debe eliminar el método **set** de la propiedad.

8. Los métodos son las operaciones que puede realizar un objeto.

   ◉ Verdadero ○ Falso

   R/Verdadero: Los Métodos son las operaciones que definen el comportamiento de un objeto.

9. El constructor solo sirve para definir el tipo de objeto que se va a crear.

   ◉ Verdadero ○ Falso

   R/Falso: El constructor permite crear la instancia del objeto que se va a crear y facilita la inicialización de las variables miembro o atributos del objeto.

10. Encapsulamiento es la particularidad que tienen los objetos de proteger los miembros privados.

    ◉ Verdadero ○ Falso

    R/Verdadero: Mediante el encapsulamiento se protegen los atributos del objeto y no se permite su manipulación desde el exterior.

11. La Herencia es la particularidad que tienen los objetos para pasar los campos privados como públicos a las clases hijas.

    ◉ Verdadero ○ Falso

    R/Falso: La Herencia define la relación entre dos clases u objetos, pero no pasan los campos internos de la clase padre a la clase hija, sino que los hereda y los puede utilizar.

12. El polimorfismo se da siempre dentro de una sola clase, mientras que la sobre carga se da entre clases distintas.

    ◉ Verdadero ○ Falso

    R/Falso: La sobrecarga se da solamente dentro de la misma clase y el polimorfismo se da entre varias clases distintas, pero con el mismo nombre del método.