**Manual de prácticas de laboratorio**

**FCI**

**materia: Programación orientada a objetos**

**Integrantes:**

**Jesús Alejandro Flores Hernández**Índice

[Objetivo. 3](#_Toc92725502)

[Introducción. 3](#_Toc92725503)

[Lenguaje. 4](#_Toc92725504)

[Compilador. 5](#_Toc92725505)

[Compilación con Programer’s Note Pad (PN). 5](#_Toc92725506)

[Ejercicios Clases e instancias 13](#_Toc92725507)

[Ejercicio 1. Clase persona. 14](#_Toc92725508)

[Ejercicio 2: Agregar atributos. 17](#_Toc92725509)

[Ejercicio 3. Clase profesor 17](#_Toc92725510)

[Ejercicio 4. Clase grupo. 18](#_Toc92725511)

[Ejercicios Property set/get. 19](#_Toc92725512)

[Ejercicios 5 set/get ejemplo, 20](#_Toc92725513)

[Ejercicios Constructores. 22](#_Toc92725514)

[Ejercicios 6 Constructores. 22](#_Toc92725515)

[Ejercicios Constructores parametrizados. 23](#_Toc92725516)

[Ejercicios 7 Constructor parametrizado. 23](#_Toc92725517)

[Ejercicios Sobrecarga de constructores. 25](#_Toc92725518)

[Ejercicios 8 Sobrecarga. 25](#_Toc92725519)

[Ejercicios Herencia 26](#_Toc92725520)

[Ejercicios 9 Herencia. 26](#_Toc92725521)

[Ejercicios Atributos protegidos 29](#_Toc92725522)

[Ejercicio 10 Atributos protegidos. 29](#_Toc92725523)

# Objetivo.

Desarrollar en el alumno la habilidad de codificación de clases y objetos, y el uso de los mismos en la resolución de problemas susceptibles de ser programados.

# Introducción.

Evolución de objetos:

Las ideas básicas sobre los objetos, nacen en los años 60’s en la universidad de **Noruega** donde un equipo dirigido por el **Dr. Nygaard**, se dedicaba a desarrollar sistemas informáticos para realizar simulaciones de sistemas físicos. Debido a que eran programas muy complejos y el mantenimiento era muy necesario (para que el software se adaptara a nuevas necesidades), se dieron cuenta de las limitaciones de la ingeniería de software tradicional, para solucionar este problema idearon una forma de diseñar el programa paralelamente al objeto físico, donde cada componente del objeto físico se correspondía con un componente de software, con lo que se simplificaba el programa y por tanto el mantenimiento exigía menor esfuerzo. Lo anterior atrajo otro beneficio muy importante que es la reusabilidad del código, echo que por si mismo repercute en una baja en el costo del software y en el tiempo requerido para el desarrollo de sistemas. La reusabilidad en la ingeniería de software no orientada a objetos, es del 5% al 15%, en cambio con los conceptos de objetos, se logra una reusabilidad del 80% o mas. Para instrumentar esas ideas diseñaron el lenguaje SIMULA-67.

En la década de los 70 **XEROX** en sus laboratorios de Palo Alto desarrolla **SMALL-TALK**.

En los años 80’s tomando ideas de Simula y de Small-Talk, en los laboratorios Bell de ATT, Stroustrup crea el lenguaje C++ como sucesor del Lenguaje C, y a este se debe la gran extensión de los conceptos de objetos.

En el área de la inteligencia artificial, se desarrolla Clos, una variante de Lisp orientada a objetos.

En sistemas operativos, el **Next-Step** de **Sun** es un S. O. Orientado a objetos. Microsoft trabaja en Cairo, IBM y Apple trabajan en Pink, como sistemas operativos que incluyen conceptos de objetos.

En las bases de datos, tenemos al **SNAP** (Strategic Networed Applications Plataform), conocido en Español como Sistemas distribuidos en línea orientados a objetos.

La Sun crea el **Java**.

Con lo anteriormente expuesto, se hace notar que los conceptos de objetos están entrando en todos los ámbitos de la computación, y de la misma manera que ha sido inevitable aprender programación estructurada o bases de datos relacionales, ahora se hace necesario aprender programación orientada a objetos, sin embargo una persona con experiencia en programación, requiere una formación de un año para manejar los conceptos de análisis, diseño y programación con objetos, además de otro año de práctica para tomar experiencia en el manejo de los mismos y un año mas para construirse sus librerías de clases, para llegar a ser un programador de objetos de alto rendimiento. Cabe hacer notar que en el mercado existen bibliotecas de clases y, también varios lenguajes traen bibliotecas de clases que le permiten al programador, realizar ciertas tareas sin tener que programarlas; como ejemplo de estos lenguajes tenemos a:

* Clipper que tiene objetos ya creados como el Tbrowse que permite el manejo de tablas.
* Visual Basic al igual que Delphi tienen objetos como botones o cuadros de dialogo con los que permite desarrollar interfaces de usuario con un mínimo de programación.

# Lenguaje.

Existen numerosos lenguajes que soportan la POO de entre los más populares tenemos Java, C++, C#, Object Pascal, Ruby y actualmente Visual Basic, la elección del lenguaje es importante en el sentido que: si bien se puede seleccionar cualquiera (o casi) para aprender a programa objetos, hay que tomar en cuenta que la curva de aprendizaje para un nuevo lenguaje, aun conociendo otro, puede ser larga por lo que debe considerarse los lenguajes a que el alumno se enfrentara en su vida profesional y que este lenguaje implemente los conceptos de POO de manera clara, esta es una tarea difícil pues puede caer el punto de la controversia, sin embargo hay que elegir uno y en este trabajo se utiliza VB NET, las razones son las siguientes:

* El SO Windows trae el compilador de VB, por lo que no requiere instalación.
* Existen varios entornos de trabajo libres para él (Framework NET, Core Net entre otros).
* Es un lenguaje muy utilizados en el mundo.
* Hay una alta probabilidad de que le sea útil al egresado.
* Es orientado a objetos.
* Sus aplicaciones son rápidas.
* Es multiparadigma.
* Es multiplataforma.
* Es fácil de aprender.
* Reduce los errores al programar.

Por esta razone es el lenguaje sobre el que se trabaja en este manual. En este manual, sin embargo, algunos ejercicios se muestran también en el lenguaje C#. El lenguaje C# tiene la misma sintaxis de C, programadores de C o Java, se sentirían cómodos con C#, aunado a que el lenguaje C# agrega algunas cosas que C y Java no tienen, como el tipo string.

**Nota**: En este material se mostrarán ejemplos en VB, que es el lenguaje para este curso, no obstante, algunos ejemplos se muestran también en C# y en java, si alguien aprendió a programar en C o en Java le servirá de referencia, sin embargo, hacer los ejercicios en estos dos últimos lenguajes es opcional. En el siguiente tema aprenderemos como configurar un editor de código para trabajar con VB, de forma similar lo puede configurar para trabajar con C# y en el material de apoyo del curso, encontrará una presentación para instalar el JDK de java y configurar el editor para trabajar con java.

# Compilador.

El compilador viene junto con la máquina virtual de NET a la que Microsoft le llama ***Framework***, no requiere instalar nada, el ***Framework NET*** viene instalado, el compilador de VB se encuentra en la carpeta del ***Framework***, vaya a ***C:\WINDOWS\Microsoft.NET***, ahí vienen dos carpetas Framework y Framework64 dependiendo donde quiera que su aplicación corra elija una de las 2. Por ejemplo, si su programa, cuando lo entregue a su cliente deberá correr en equipos de 32 bits, elija el compilador que viene en la carpeta Framework, si deberá correr en equipos de 64 bits elija la carpeta Framework64. Dentro de esas carpetas viene el compilador de Visual Basic “vbc.exe” y el de C# “csc.exe”.

Una vez ubicado el compilador, debemos seleccionar el ambiente de desarrollo, que nos facilitará la creación y edición de código, aquí trabajaremos con aplicaciones de consola, para este tipo de aplicaciones puede usar varios editores o ambientes de trabajo: *Sublime*, *Visual Studio Code*, *Sharp Develop*. Debido a su poco peso, fácil configuración y que tiene una versión portable, el editor que se usará en este curso es el ***Programer’s Note Pad (PN)***, puede instalar si lo desea el ***Sharp Develop*** versión 3.x o 4.4, El ***Sharp Develop*** le permite crear aplicaciones de consola y de aplicaciones visuales también.

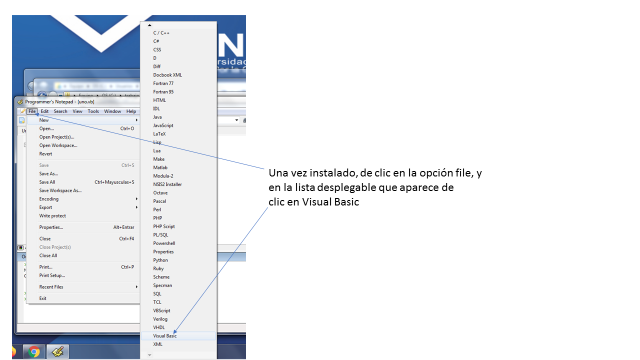
# Compilación con Programer’s Note Pad (PN).

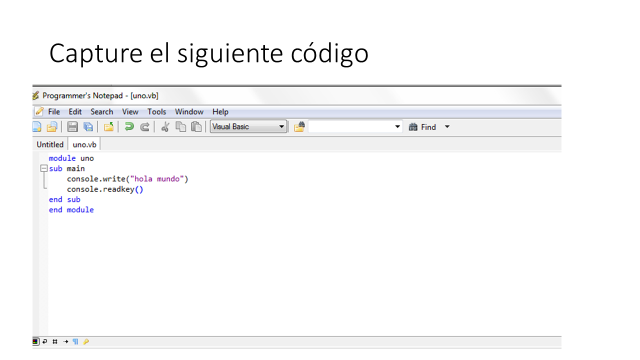
Instale el PN desde : <https://www.pnotepad.org/download/>

Ahí tiene una versión para instalar y una versión portable, escoja la que guste y a continuación:

* Instale si eligió la versión instalable, busque en el escritorio el logo del PN o en el botón de inicio y ejecute.
* Si eligió la versión portable, le descargará un archivo comprimido (zip), desempaquete en una carpeta, en esa carpeta encontrará el archivo pn.exe, ejecútelo.

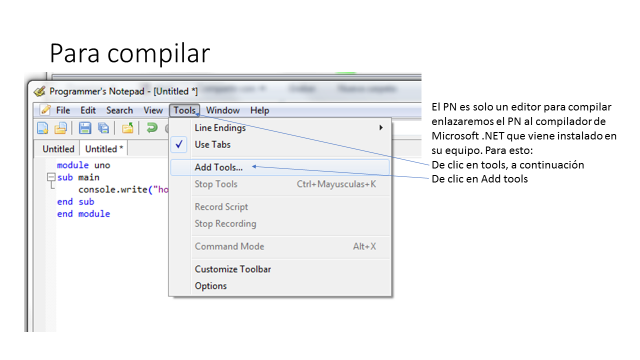
Una vez instalado el PN, deberá configurarlo para que desde el mismo editor compile sus programas esto lo hace de la siguiente manera (en el material de apoyo se muestra una presentación para configurar el PN para compilar y ejecutar código de VB, aquí se muestra solo un resumen de eso, le recomiendo que vea la presentación, que es un documento más completo para la configuración del PN). Para configurar PN, inicie la aplicación PN y a continuación:

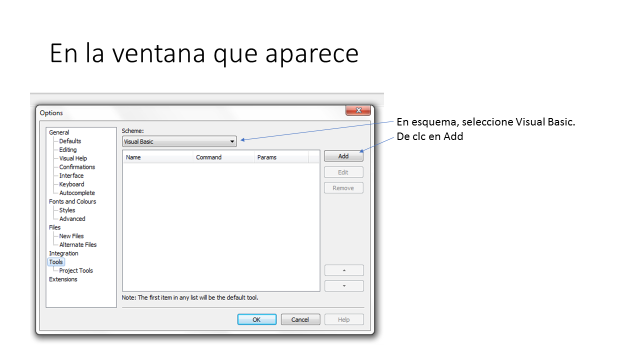


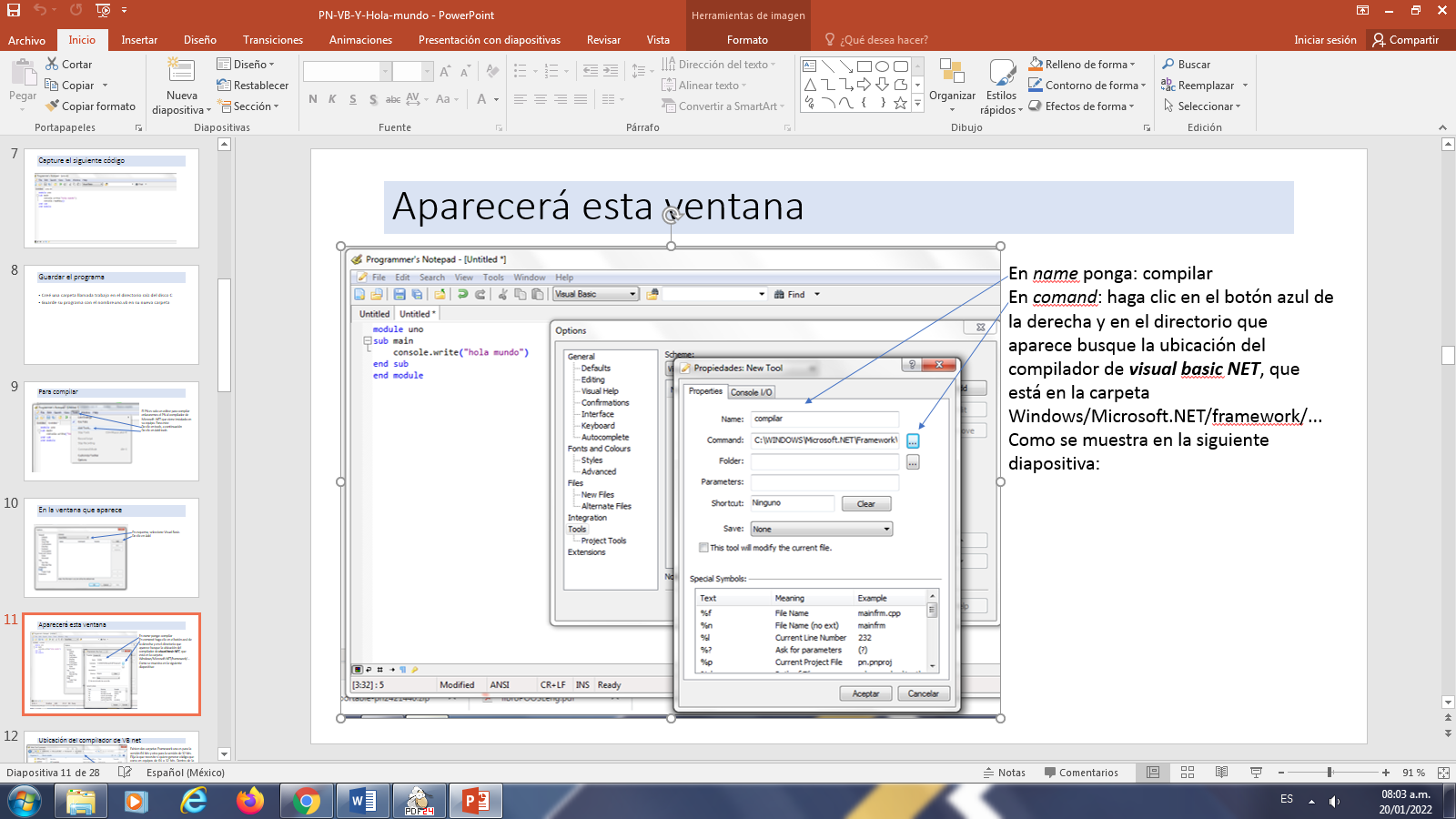


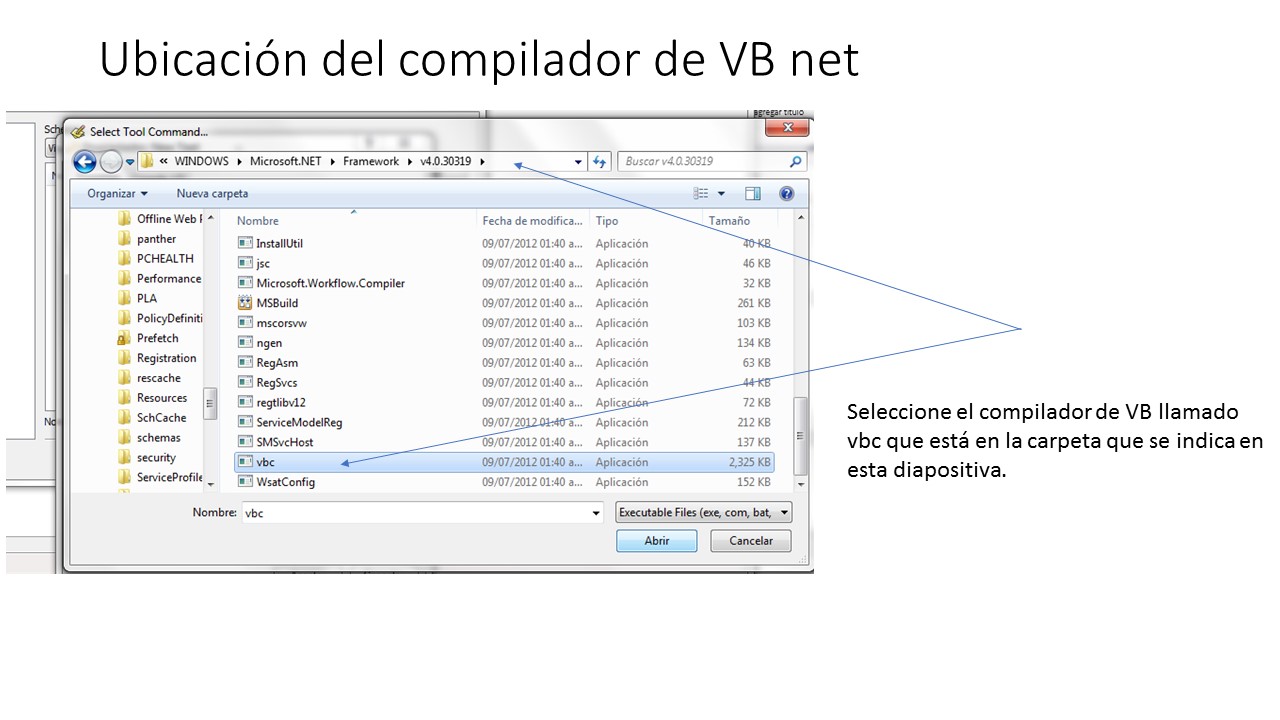
Cree una carpeta llamada ***trabajo*** en el directorio raíz del disco C

Guarde su programa con el nombre ***uno.vb*** en su nueva carpeta



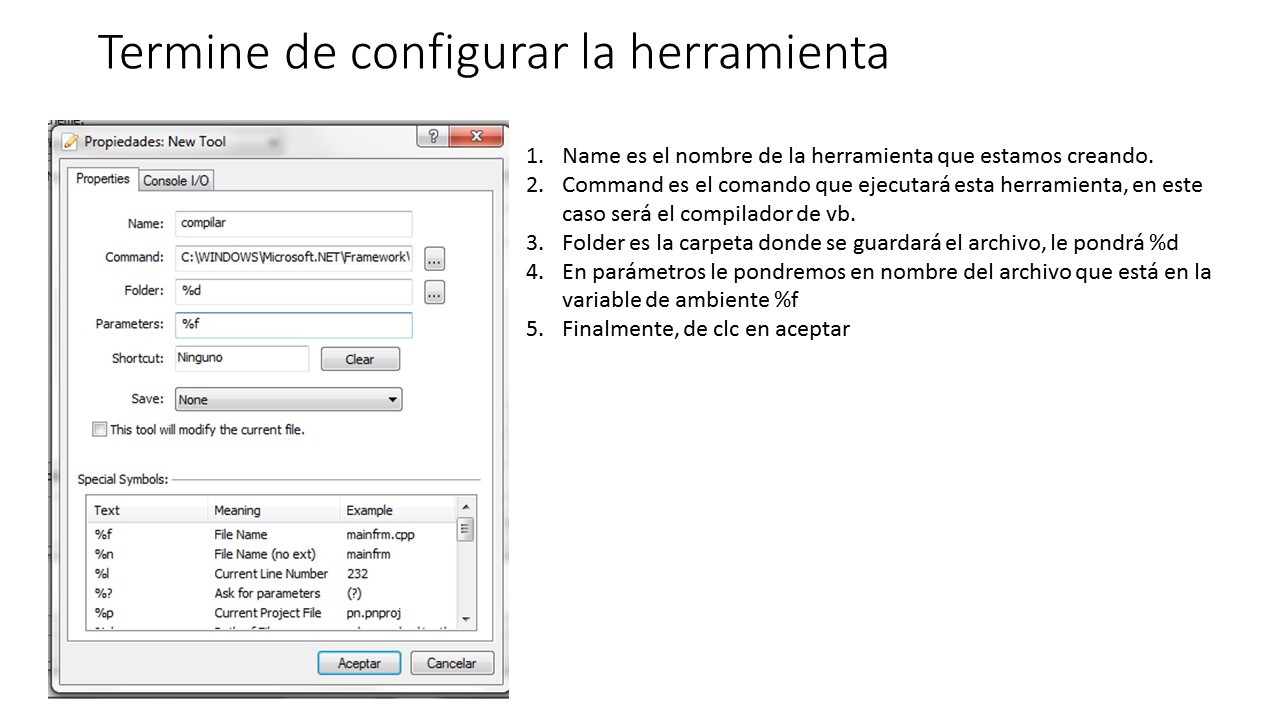


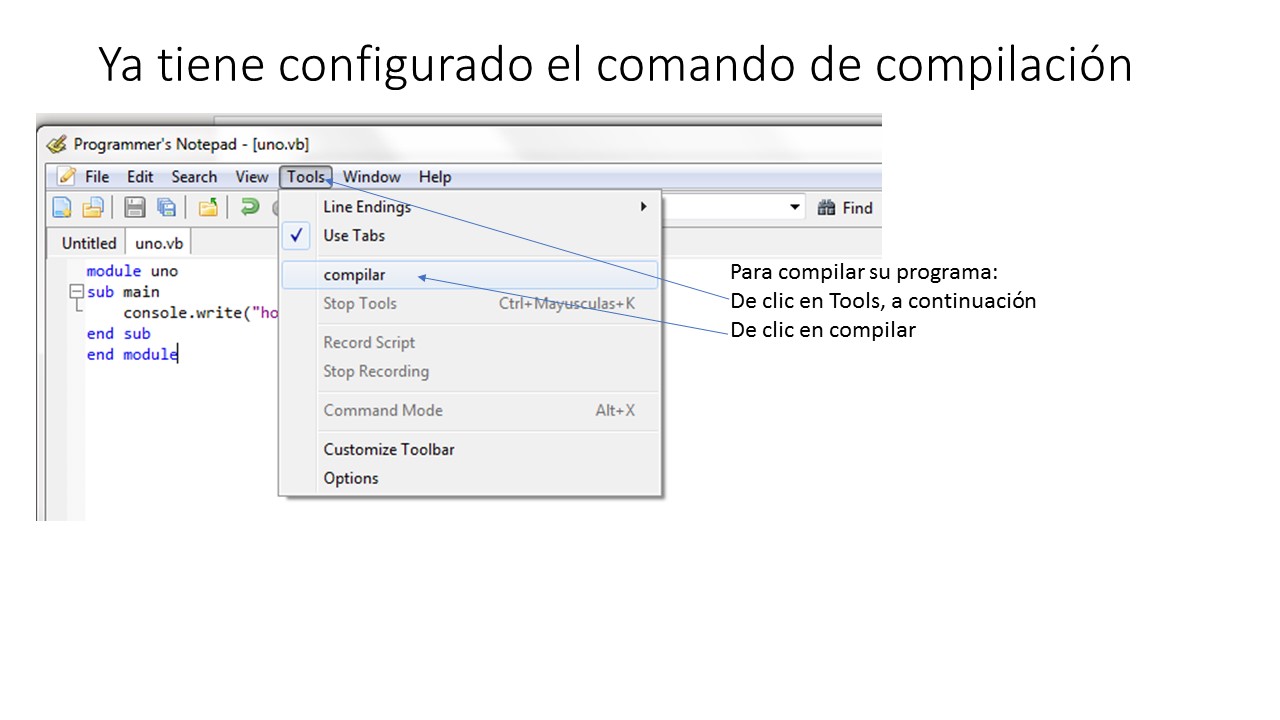




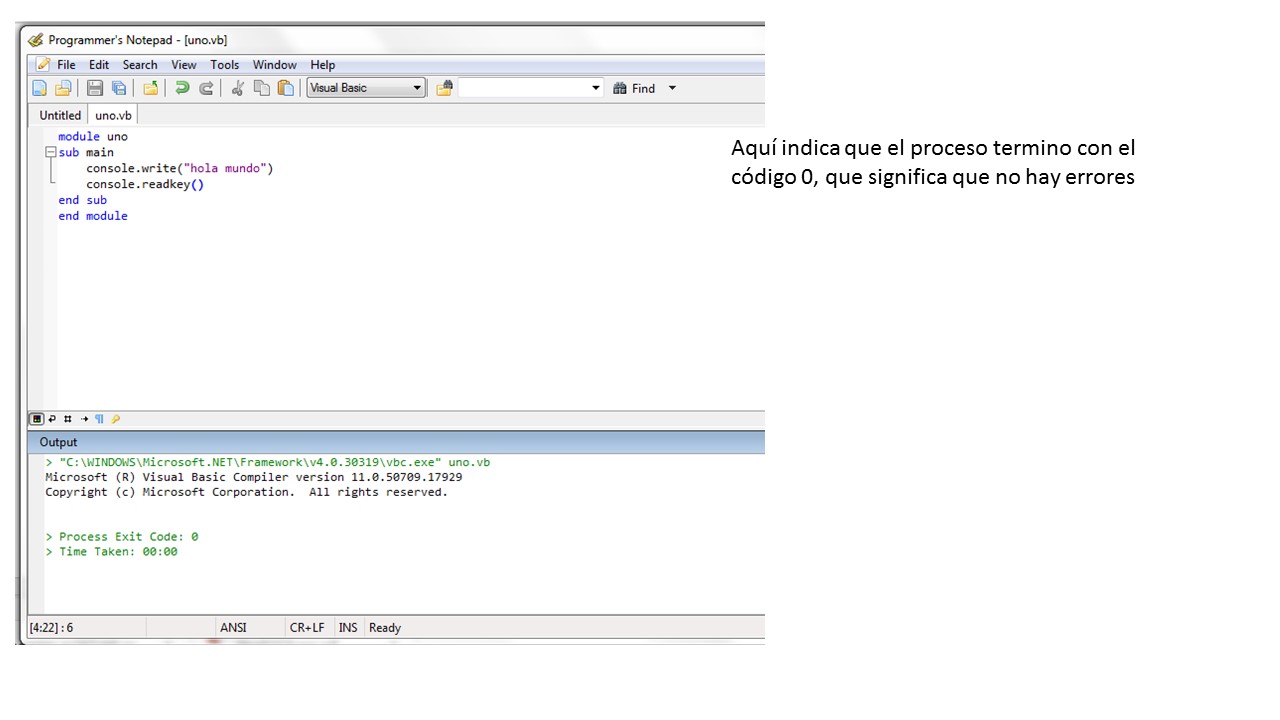
En mi equipo use la ruta: ***C:\WINDOWS\Microsoft.NET\Framework64\v4.0.30319*** .

Dentro de la carpeta Framework64 tengo varias versiones y elegí la versión 4.0.30319, de clic en Abrir.

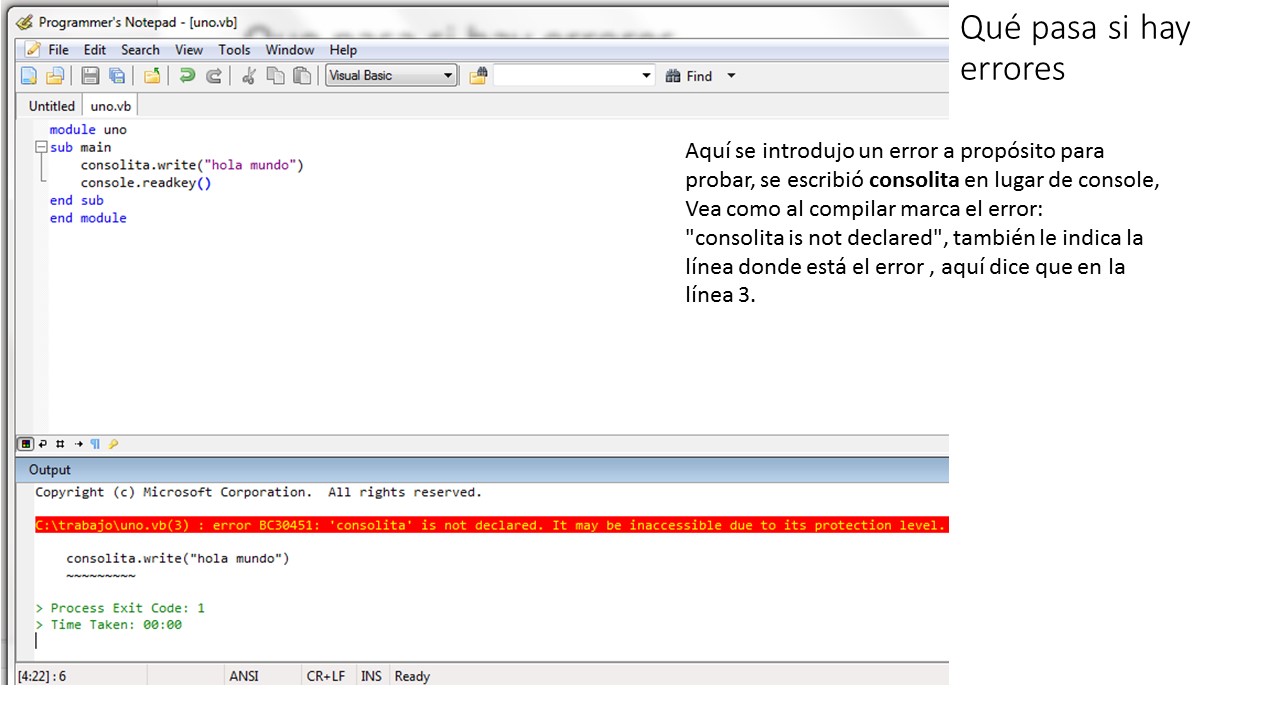




Aparecerá el resultado de su compilación en una ventana emergente llamada output, si es pequeña con el ratón la puede redimensionar y arrastrar a la parte baja de su ventana del editor para anclarla. Quedará así:

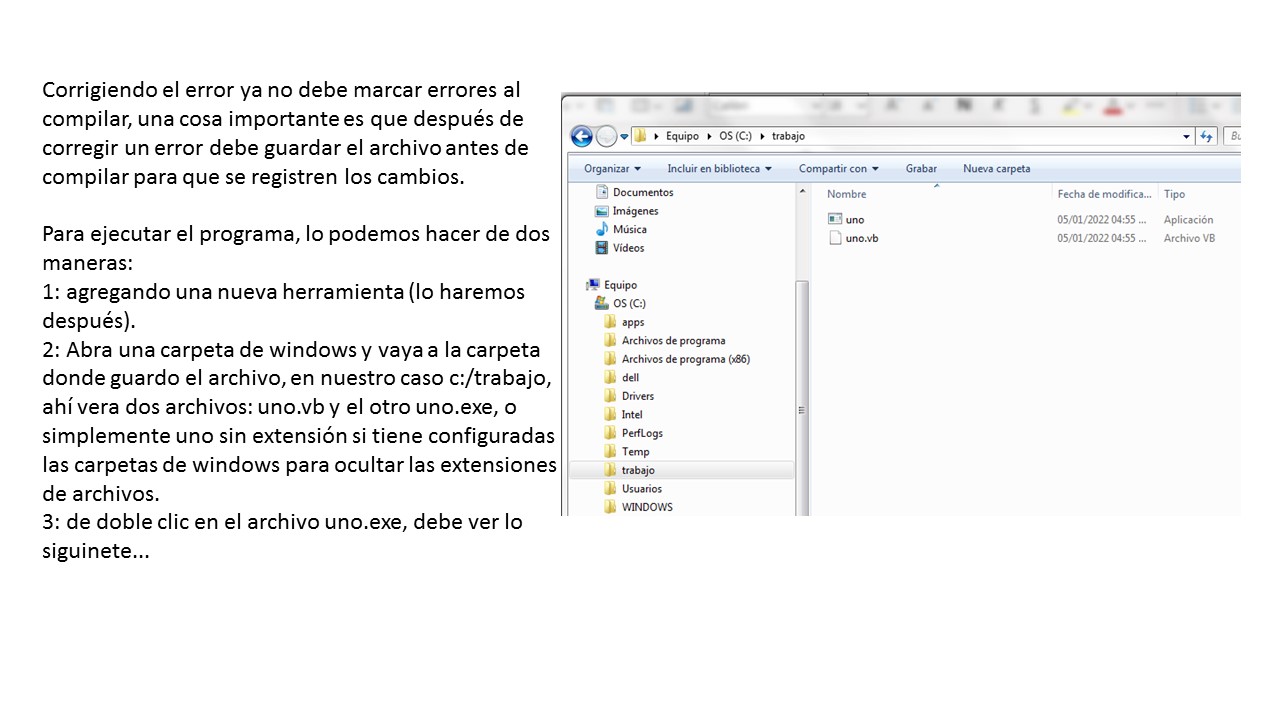


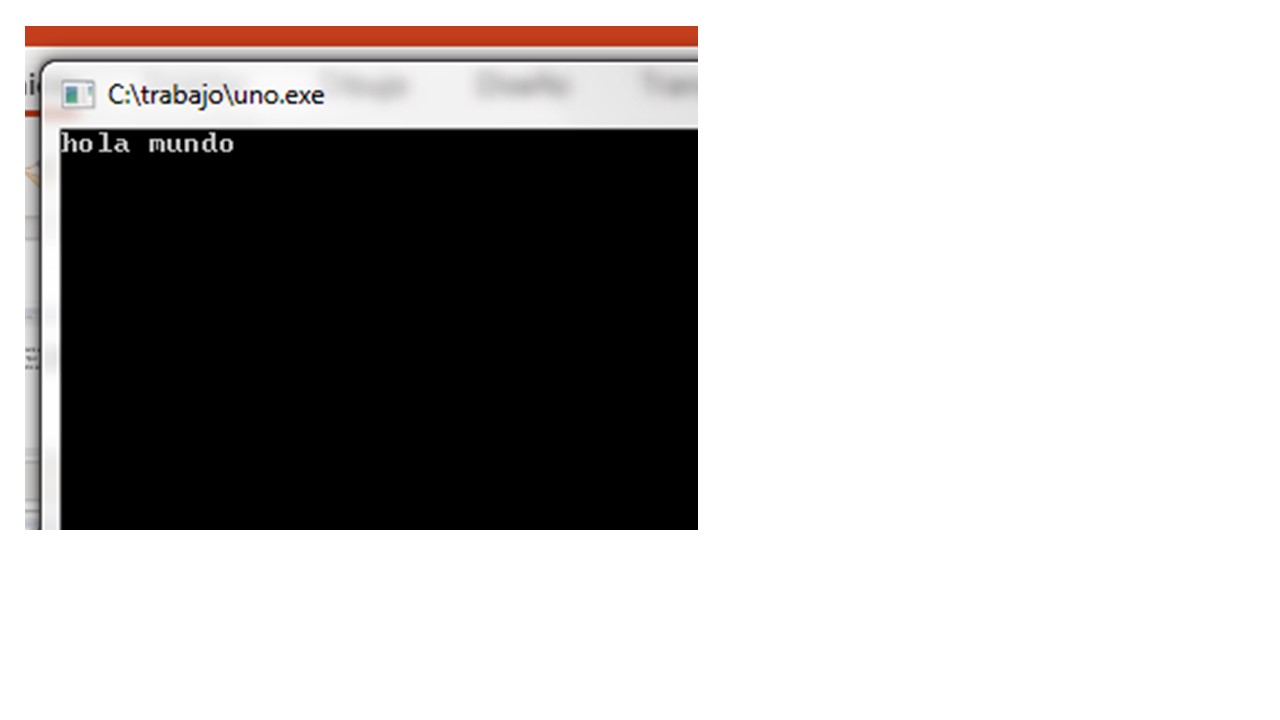
¿Qué pasa si hay errores?

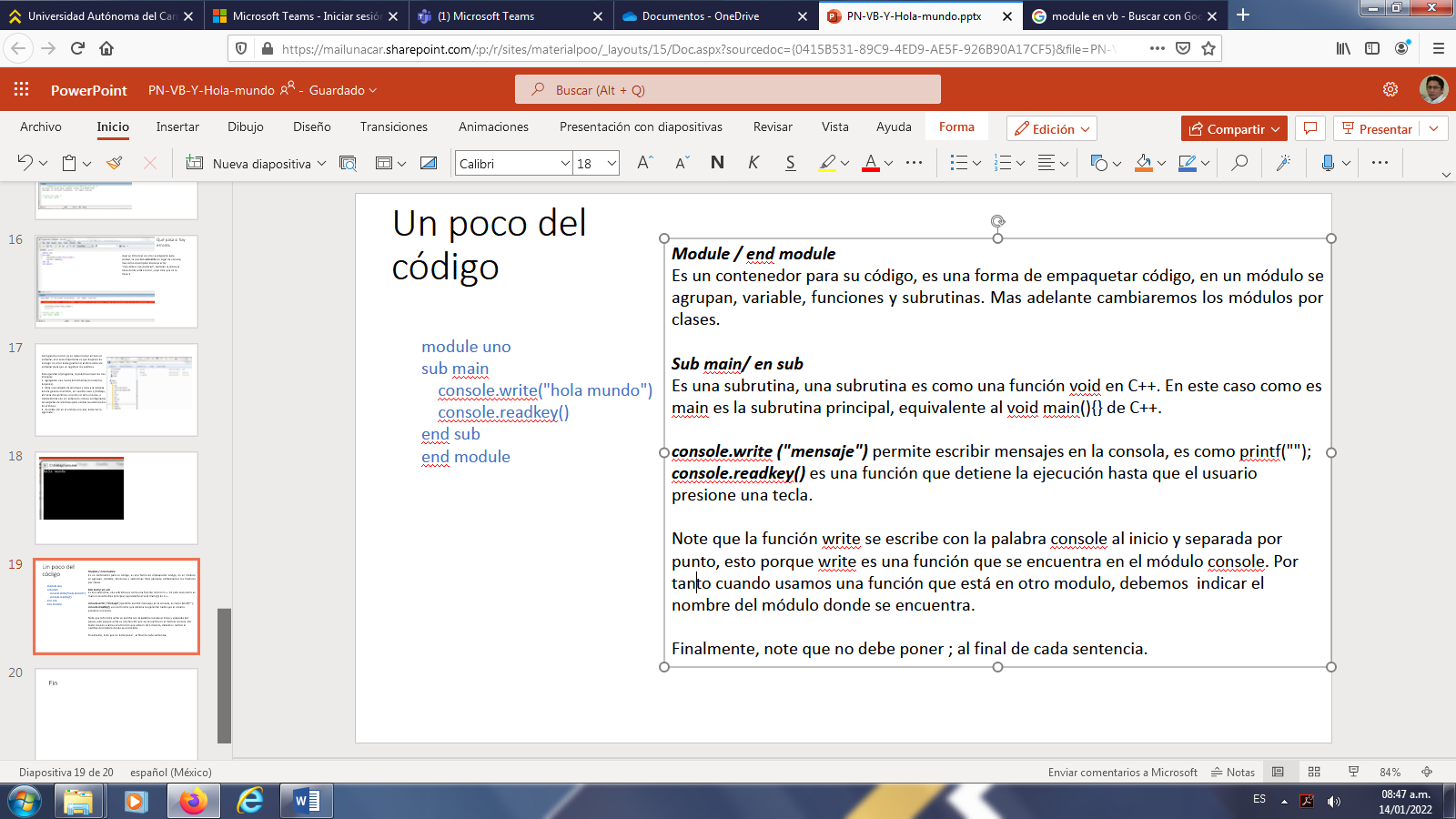


En la parte baja izquierda de su ventana de edición en PN vera el símbolo “***#***”, haga clic en ese símbolo y las líneas de código aparecerán numeradas.

Corrigiendo el error en caso de que haya y compilando, se tiene un archivo ejecutable.







De manera similar puede configurar el PN para compilar en C#, para esto deberá poner a sus archivos la extensión .cs y configurar una herramienta (tool) para C# de manera similar a como configuro para VB, excepto que elegirá el compilador de C# llamado csc.exe y en el esquema elegirá C#.

En el caso de la versión portable, una vez configurado el PN, puede tomar la carpeta completa que descomprimió copiarla a un ***usb*** y cuando quiera usar el PN en cualquier equipo, solo tendrá que copiar esta carpeta, y ejecutar el archivo ***pn.exe*** dentro de ella, ya estará configurado.

Ejemplo “hola mundo con C#”

|  |
| --- |
| Archivo: hola\_mundo.cs |
| using System;  namespace ejemplos{  class prueba{  public static void Main(){  Console.Write("hola mundo");  Console.ReadKey();  }  }  } |

Algunas precisiones:

Primero: Las funciones de biblioteca de NET, se agrupan en clases y las clases se agrupan en espacios de nombres, System es el espacio de nombres que contiene los tipos fundamentales de NET y contiene también otros espacios de nombres (piénselo como una biblioteca de funciones, excepto que esas funciones vienen dentro de una clase y la clase en un espacio de nombres), por ejemplo, las funciones Write, WriteLine, ReadKey, se encuentran dentro de la clase Console y la clase Console dentro del espacio de nombres System, de manera que para poder usar esas funciones hay que cargar el espacio de nombres System e indicar en el llamado a la función, la clase donde se encuentran:

***using System;***

En el caso de VB el espacio de nombres system se cargaría así:

***Imports system***

Excepto que en VB no es necesario hacerlo explícitamente por que el compilador carga el espacio de nombres ***system*** por default. En el caso de C# hay que cargarlo explícitamente.

Segundo: Cuando programa en C# debe elegir un espacio de nombres que albergará su código, por eso la línea:

namespace ejemplos{}

en este caso ejemplos es el espacio de nombres que contendrá su código.

Para VB esto es opcional, y puede, si lo desea elegir su espacio de nombres así:

namespace miespacio

end namespace

|  |
| --- |
| Archivo: hola\_munod.vb |
| imports system  namespace ejemplos  public class mensaje  public shared sub main  console.writeline("hola mundo")  console.readkey()  end sub  end class  end namespace |

Este ejemplo en Vb funciona también sin cargar explícitamente system ni definir un espacio de nombres, así:

|  |
| --- |
| public class mensaje  public shared sub main  console.writeline("hola mundo")  console.readkey()  end sub  end class |

Finalmente, en estos dos ejemplos estamos usando una función ***main*** declarada como estática: ***static*** en C#, ***shared*** en VB, las funciones estáticas son funciones de clase y no de objetos, pero esto se entenderá más adelante, también podemos trabajar con una función ***main*** dentro de un código estructurado tradicional, que en VB se pone dentro de un módulo (module), En C# esto no es posible, por lo que debe hacerse siempre en un función:

***public static void main().***

Ejemplo hola nundo con java

|  |
| --- |
| Archivo: ejemplo3.java |
| import java.io.\*;  public class ejemplo{  public static void main(String args[]){  Console cnsl=System.console();  System.out.println("hola mundo");  cnsl.readLine("presione enter para continuar...");  }  } |

# Ejercicios Clases e instancias

En el aprendizaje de la POO, veremos una serie de ejercicios que nos ayudarán a comprender mejor estos conceptos.

# Ejercicio 1. Clase persona.

Escriba un código que defina una clase para objetos del tipo persona con dos atributos (nombre y edad, por ejemplo):

Solución:

|  |
| --- |
| Archivo:persona.vb |
| class persona  private nombre as string  private edad as string  public sub setNombre(nombre as string)  me.nombre=nombre  end sub  public function getNombre as string  return me.nombre  end function  end class  module prueba  sub main  dim p as new persona  p.setNombre("juan")  console.writeline(p.getNombre())  console.readkey()  end sub  end module |

O con la subrutina main() dentro de la clase así:

|  |
| --- |
| Archivo:persona2.vb |
| class persona  private nombre as string  private edad as string  public sub setNombre(nombre as string)  me.nombre=nombre  end sub  public function getNombre as string  return me.nombre  end function  public shared sub main  dim p as new persona  p.setNombre("juan")  console.writeline(p.getNombre())  console.readkey()  end sub  end class |

Explicación: Las clases definen un tipo de objetos, pero no son el objeto. Para trabajar con objetos de una clase hay que instanciarlos(crearlos) y en el ejemplo lo hacemos dentro del módulo prueba. Cuando tenemos una clase, sus atributos y métodos no existen como tal, existen hasta que creamos instancias de esa clase así:

***dim p as new persona***

por tanto, los métodos definidos en una clase estarán asociados a los objetos (instancias) de esa clase y solo pueden ser llamados desde un objeto así:

***p.getNombre()***

Por otro lado, para que un programa funcione debe tener una función main(), para eso definimos un módulo con una función main(), desde la que creamos el objeto y llamamos a sus métodos, el modulo al no ser una clase no requiere tener instancias, es programación estructurada normal, pero recuerde es muy importante que la función main() exista, porque el sistema operativo empezará la ejecución de cualquier programa buscando la función main(), si esta no existe su programa no hará nada, ni compilara siquiera, para el ejemplo de poner la función main dentro de la clase como función estática, se explicará cuando veamos las funciones estáticas.

# Ejercicio 1.1: Clase persona en C#

|  |
| --- |
| Archivo: persona.cs |
| using System;  namespace basicos{  class persona{  private string Nombre;  private int Edad;  public void setNombre (string nombre){  this.Nombre=nombre;  }  public string getNombre (){  return this.Nombre;  }  public void setEdad (int edad){  this.Edad=edad;  }  public int getEdad (){  return this.Edad;  }  public static void Main(string[] args){  persona p=new persona();  Console.WriteLine("Prueba de clases");  p.setNombre("Andrea");  Console.WriteLine("mi nombre es: "+p.getNombre());  Console.ReadKey();  }  }  } |

Aquí se está cambiando el lenguaje y cómo ve es muy parecido a Java o C++, usa la misma sintaxis y también hace diferencia entre mayúsculas y minúsculas, si desea probar este código, captúrelo en PN, el compilador de C# viene también con el Framework y se llama “csc.exe”, puede compilar programas en C# agregando una herramienta en PN y ligándola al compilador de C# de manera similar que lo hicimos en VB, solo recuerde en esquema, elegir C#.

Clase persona en java:

|  |
| --- |
| Archivo: persona.java |
| import java.io.\*;  public class persona{  private String nombre;  private int edad;  public void setNombre(String nombre){  this.nombre=nombre;  }  public String getNombre(){  return this.nombre;  }  public void setEdad(int edad){  this.edad=edad;  }  public int getEdad(){  return this.edad;  }  public static void main(String args[]){  Console cnsl=System.console();  persona p=new persona();  p.setNombre("juan");  p.setEdad(24);  System.out.println("nombre:"+p.getNombre());  cnsl.readLine("presione enter para continuar...");  }  } |

# Ejercicio 2: Agregar atributos.

Agregue el código necesario para alimentar y obtener el dato del atributo edad en la clase persona.

Modifique la clase ejemplo01:

# Ejercicio 3. Clase profesor

Escriba un código que defina objetos de la clase profesor con tres atributos (pej: Número de empleado, nombre y carrera).

Solución:

|  |
| --- |
| Archivo:profesor.vb |
| class profesor  private numEmpleado as string  private nombre as string  private carrera as string  public sub setNumEmpleado(ne as string)  me.numEmpleado=ne  end sub  public function getNumEmpleado as string  return me.numEmpleado  end function  public sub setNombre(nombre as string)  me.nombre=nombre  end sub  public function getNombre as string  return me.nombre  end function  public sub setCarrera(carrera as string)  me.carrera=carrera  end sub  public function getCarrera as string  return me.carrera  end function  end class  module prueba  sub main  dim p as new profesor  p.setCarrera("LTI")  console.writeline(p.getCarrera())  console.readkey()  end sub  end module |

**Agregación**

La agregación es un mecanismo por el cual un objeto es incrustado en otro, ejemplos de agregación los tenemos en el objeto motor, dentro del objeto auto, o el objeto alumno dentro del objeto grupo.

# Ejercicio 4. Clase grupo.

Diseñe la clase grupo con 3 atributos.

Además de los atributos deberá manejar un conjunto de alumnos agregados al grupo.

Solución:

|  |
| --- |
| Archivo: grupo.vb |
| class alumno  private nombre as string  public sub setNombre(nombre as string)  me.nombre=nombre  end sub  public function getNombre()  return nombre  end function  end class  class grupo  private numeroGrupo as string  private ubicacion as string  private capacidad as string  private numAlumnos as integer=0  private listaAlumnos(10) as alumno  public sub agregaAlumno(a as alumno)  me.listaAlumnos(me.numAlumnos)=a  me.numAlumnos=me.numAlumnos+1  end sub  public function devuelveAlumno() as alumno  if(me.numAlumnos>0) then  me.numAlumnos=me.numAlumnos-1  return me.listaAlumnos(me.numAlumnos)  else  return nothing  end if  end function  end class  module prueba  sub main()  dim g as new grupo  dim a as new alumno  a.setNombre("juana")  g.agregaAlumno(a)  dim b as new alumno  b.setNombre("rosa")  g.agregaAlumno(b)  dim x as alumno=g.devuelveAlumno()  console.writeline(x.getNombre())  console.readkey()  end sub  end module |

Nota: en este ejemplo no se está verificando el límite del arreglo de alumnos que es de 10, deberá modificarse el programa para verificar este dato.

# Ejercicios Property set/get.

Las propiedades (property) son una forma de controlar el acceso a los atributos de un objeto, respetando su encapsulación y permitiendo una sintaxis de acceso más intuitiva, una propiedad puede tener una sentencia set/ get o ambas y estas permiten entrar y sacar información de la propiedad. Por ejemplo, si tiene una clase persona con un atributo:

private clave as string

puede agregar una propiedad para acceder a este atributo:

***property mi\_clave***

***set (dato)***

***clave=dato***

***end set***

***get***

***return clave***

***end get***

***end property***

y ahora si tiene una instancia de la clase persona, puede acceder al atributo así:

***dim p as new persona***

***p.mi\_clave="juan"***

***console.writeline(p.mi\_clave)***

Usando el nombre de la propiedad puedo introducir o sacar la información del atributo asociado a la propiedad.

# Ejercicios 5 set/get ejemplo,

|  |
| --- |
| Archivo:sete\_get\_0.vb |
| class persona  private clave as string  private nombre as string  private edad as string  property mi\_clave  set (dato)  clave=dato  end set  get  return clave  end get  end property  end class  module prueba  sub main  dim p as new persona  p.mi\_clave="juan"  console.writeline(p.mi\_clave)  console.readkey()  end sub  end module |

Lo normal es que a las propiedades se les dé un nombre simple pues es la interfaz que verá el programador que use mi clase, y a los atributos se les da un nombre parecido por ejemplo con una subraya antes del nombre.

|  |
| --- |
| Archivo: set\_get.vb |
| class persona  private \_clave as string  private \_nombre as string  private \_edad as string  public property clave  set (dato)  \_clave=dato  end set  get  return \_clave  end get  end property  end class  module prueba  sub main  dim p as new persona  p.clave="juan"  console.writeline(p.clave)  console.readkey()  end sub  end module |

En el ejemplo anterior será fácil para usted agregar las propiedades de los demás atributos.

Dado que las propiedades y los atributos son elementos diferentes, se deben llamar diferente, pero están asociados. Una propiedad controlará el acceso a un atributo, por tanto, lo mejor es que tengan un nombre muy similar y dado que el usuario de la clase interactúa con las propiedades y no con los atributos, lo mejor es que el nombre más “fácil” de recordar o usar lo tenga la propiedad y el nombre menos “fácil” de recordar lo tenga el atributo.

|  |
| --- |
| Archivo:set\_get.cs |
| using System;  class persona{  private string Nombre;  private int Edad;  public string nombre{  set{Nombre=value;}  get{return Nombre;}  }  public int edad{  set{Edad=value;}  get{return Edad;}  }    }    public class prueba{  public static void Main(string[] args){  persona p=new persona();  Console.WriteLine("Prueba de clases");  p.nombre="Andrea";  p.edad=18;  Console.WriteLine("Nombre es: "+p.nombre+" Edad: "+p.edad);  Console.ReadKey();  }  }  } |

En el ejemplo con C# no usamos subraya en los nombres de atributos, los distinguimos por medio de la mayúscula en la primera letra del atributo, vea que el nombre de la propiedad no empieza con mayúscula, el C# al hacer diferencia entre mayúsculas y minúscula toma como identificadores diferentes a “nombre” y “Nombre”, de esa manera no hay ambigüedad en los nombres de atributo y propiedad y el compilador no tiene problemas para distinguirlos.

# Ejercicios Constructores.

Los constructores son métodos que se ejecutan automáticamente cuando un objeto es instanciado, esto puede servir para inicializar los datos del objeto. Un constructor no puede ser llamado explícitamente, se ejecutan al crear (instanciar) objetos. En VB el constructor lleva el nombre "***new***". A continuación se muestra un ejemplo de un constructor que lo que hace es “limpiar” el contenido de los atributos del objeto creado nombre y sexo asignando una cadena nula.

# Ejercicios 6 Constructores.

crear una clase persona con un constructor que limpie las variables al inicio.

|  |
| --- |
| Archivo: constructor1.vb |
| class persona  private nombre as string  private sexo as char  public sub new  nombre=""  sexo=""  end sub  public sub setNombre(m as string)  nombre=m  end sub  public sub setSexo(s as char)  sexo=s  end sub  public function getNombre as string  return nombre  end function  public function getSexo as char  return sexo  end function  end class  module prueba  sub main()  dim p as persona 'referencia a persona  p=new persona() 'crea una persona  end sub  end module |

# Ejercicios Constructores parametrizados.

Los constructores pueden tener parámetros, normalmente estos se utilizan para inicializar los valores de los atributos. Esto es útil cuando se sabe el contenido de los atributos al instancia el objeto.

# Ejercicios 7 Constructor parametrizado.

Crea una clase persona con un constructor parametrizado que inicialice sus atributos con los datos dados.

|  |
| --- |
| Archivo: constructorParam.vb |
| class persona  private nombre as string  private sexo as char  public sub new(nombre as string, sexo as char)  me.nombre=nombre  me.sexo=sexo  end sub  public sub setNombre(m as string)  nombre=m  end sub  public sub setSexo(s as char)  sexo=s  end sub  public function getNombre as string  return nombre  end function  public function getSexo as char  return sexo  end function  public function quienEres()  return "soy "+me.nombre+" y soy de sexo "+me.sexo  end function  end class  module prueba  sub main()  dim p as persona 'referencia a persona  p=new persona("Jorge Nitales", "m") 'crea una persona  console.writeline(p.quienEres())  console.readline()  end sub  end module |

Nota: Note que en el constructor de la clase persona se reciben los parámetros nombre y sexo que tienen los mismos nombres que los atributos de la clase, por esto para evitar ambigüedades se usa la notación ***me.nombre=nombre***

La sentencia ***me***. Dentro de una clase hace referencia a sus propios atributos.

En C# un constructor es una función con el mismo nombre de la clase y en caso de ***se me*** usa ***this***.

|  |
| --- |
| Archivo:constructor1.cs |
| using System;  class persona{  private string Nombre;  protected int Edad;  public persona(string n,int e){  Nombre=n;  Edad=e;  }  public string nombre{  set{Nombre=value;}  get{return Nombre;}  }  public int edad{  set{Edad=value;}  get{return Edad;}  }  public virtual void muestra(){  Console.WriteLine("Nombre es: "+Nombre+" Edad: "+Edad);  }  }  public class constructores{  public static void Main(string[] args){  alumno p=new persona("Andrea",18);  p.muestra();  Console.ReadKey();  }  } |

# Ejercicios Sobrecarga de constructores.

Una clase puede tener más de un constructor, esto permitirá hacer inicializaciones distintas de un objeto dependiendo de nuestras necesidades, el compilador distingue los diferentes constructores por el número y tipo de parámetros enviados.

# Ejercicios 8 Sobrecarga.

Crea una clase alumno con 2 constructores, uno que limpie los valores de los atributos y otro que inicialice con los valores dados.

|  |
| --- |
| Archivo: constructorSobre.vb |
| class persona  private nombre as string  private sexo as char  public sub new()  nombre=""  sexo=""  end sub  public sub new(n as String, s as Char )  nombre=n  sexo=s  end sub  public sub setNombre(m as string)  nombre=m  end sub  public sub setSexo(s as char)  sexo=s  end sub  public function getNombre as string  return nombre  end function  public function getSexo as char  return sexo  end function  end class  module prueba  sub main()  'se usa el constructor con parametros  dim p as persona=new persona("Andrea","F")  'se usa el constructor sin parametros  dim q as persona=new persona()  q.setNombre("Alejandra")    end sub  end module |

# Ejercicios Herencia

La herencia nos permite extender las características de una clase aprovechando la programación existente para otra clase, por ejemplo, si tenemos definida una clase persona con ciertas propiedades (atributos y métodos), y ahora deseamos una clase alumno, notamos que un alumno es una persona y por lo tanto puedo ***heredar*** lo que ya tenía programado para la persona y agregar solo lo que a la persona le falta para ser un alumno, ejemplo:

# Ejercicios 9 Herencia.

crea una clase alumno que herede de la clase persona que se usó previamente, la clase alumno debe heredar las propiedades de la persona y agregar la propiedad matrícula.

|  |
| --- |
| Archivo: alumno\_herencia.vb |
| class persona  private nombre as string  private sexo as char  public sub new()  nombre=""  sexo=""  end sub  public sub new(n as String, s as Char )  nombre=n  sexo=s  end sub  public sub setNombre(m as string)  nombre=m  end sub  public sub setSexo(s as char)  sexo=s  end sub  public function getNombre as string  return nombre  end function  public function getSexo as char  return sexo  end function  end class  class alumno  inherits persona 'hereda de la clase persona  private matricula as string  private semestre as integer  public sub new()  matricula=""  semestre=0  end sub  public sub new(nom as string, sex as char,mat as string, sem as integer)  mybase.new(nom,sex)  matricula=mat  semestre=sem  end sub  public sub setMatricula(m as string)  matricula=m  end sub  public sub setSemestre(s as integer)  semestre=s  end sub  public function getMatricula as string  return matricula  end function  public function getSemestre as integer  return semestre  end function  end class  module prueba  sub main()  'se usa el constructor con parametros  dim p as alumno=new alumno("Andrea","F","1234",3)  'se usa el constructor sin parametros  dim q as alumno=new alumno()  q.setNombre("Alejandra")    end sub  end module |

Hay algunas cosas que puntualizar:

Primero: para heredar, al declarar una clase x que hereda de una clase y, debajo de la primera línea de la clase derivada, se coloca la palabra: inherits y. Que indica que hereda de la clase y.

Segundo: cuando en una clase base (la clase padre) se tiene un constructor parametrizado, este constructor de la clase base se puede usar desde la clase derivada usando la sentencia mybase.new(parámetros).

Ejemplo en C#:

|  |
| --- |
| Archivo: herencia.cs |
| namespace basicos{  using System;  class persona{  private string Nombre;  private int Edad;  public persona(string Nombre, int Edad){  this.Nombre=nombre;  this.Edad=Edad;  }  public string nombre{  set{Nombre=value;}  get{return Nombre;}  }  public int edad{  set{Edad=value;}  get{return Edad;}  }    }  class alumno:persona{  private string Carrera;  public alumno(string Nombre, int Edad, string Carrera):base(Nombre,Edad){  this.Carrera=Carrera;  }  public string carrera{  set{Carrera=value;}  get{return Carrera;}  }  }  public class holaMundo{  public static void Main(string[] args){  alumno p=new alumno("Rosa",34,"ISC");  Console.WriteLine("Prueba de clases");  Console.WriteLine("Nombre es: "+p.nombre+" Edad: "+p.edad+" Carrera "+p.carrera);  Console.ReadKey();  }  }  } |

En C# la herencia se implmenta así:

***class alumno:persona{}***

poniendo dos puntos y enseguida el nombre de la clase padre.

La llamada a un constructor de la clase padre es:

***public alumno(string Nombre, int Edad, string Carrera):base(Nombre,Edad){}***

en el constructor de la clase derivada se pone en la cabecera de la función

***:base(parámetros si son necesarios)***

Así estamos indicando que antes de ejecutar su código, el constructor de la clase derivada debe ejecutar el constructor de la clase base, y enviamos los parámetros si son necesarios.

# Ejercicios Atributos protegidos

Los atributos de una clase hasta ahora podían ser públicos o privados, nunca los hemos dejado públicos porque eso atenta contra la encapsulación, que nos dice que los atributos de un objeto solo deben ser accedidos directamente desde dentro de su objeto mismo, violar esta regla nos regresaría a la programación estructurada de la que originalmente tratamos de escapar. ¿Pero y si necesitamos accederlos?, una buena programación hará que nunca necesite accederlos, ejemplo:

# Ejercicio 10 Atributos protegidos.

Escriba un programa que tenga una clase alumno que herede de persona y después escriba una función que devuelva en una cadena la información del alumno, separada por comas.

|  |
| --- |
| Archivo: protected0.vb |
| class persona  private nombre as string  private edad as string  sub new(nombre as string, edad as integer)  me.nombre=nombre  me.edad=edad  end sub  'ponga aqui las propiedades para acceder a nombre y edad  end class  class alumno  inherits persona  private carrera as string  sub new(nombre as string, edad as integer, carrera as string)  mybase.new(nombre,edad)  me.carrera=carrera  end sub  function traeDatos()  return "soy "+nombre+" mi clave es:"+clave+" y mi edad es:"+edad+" y mi carrera es:"+carrera  end function  end class  module prueba  sub main  dim p as new alumno("juan",23,"ICO")  console.writeline(p.traeDatos())  console.readkey()  end sub  end module |

Todo parece correcto excepto que: Marca error, aquí:

return "soy "+nombre+" mi clave es:"+clave+" y mi edad es:"+edad+" y mi carrera es:"+carrera

Esto debido a que como los atributos de la clase persona son privados, no puedo accederlos desde un objeto de la clase alumno, aunque sea una clase derivada (clase hija).

¿Cómo resuelvo esto?: hay dos formas la primera es:

1. Cambie la declaración ***private*** de los atributos nombre y edad y declárelos como ***protected***:
2. Usar polimorfismo.

Aquí usaremos la solución 1, declarando los atributos así:

***protected nombre as string***

***protected edad as string***

Cambie estas dos líneas en el archivo de arriba y pruebe el código.

La propiedad ***protected*** de un atributo sigue haciendo que los atributos sean privados al resto del código, excepto a los descendientes de la clase, todas las clases que hereden de una clase con algún atributo declarado como ***protected*** tendrán acceso a este.

# Ejercicios Polimorfismo.

Suponga que tiene una clase persona con atributos clave, nombre y edad, un constructor y dos métodos un método traeDatos que devuelve la información de los atributos y el método mensaje que muestra el mensaje hola y llama al método trae datos para presentarse.

|  |
| --- |
| Archivo: saludo.vb |
| class persona  private clave as string  private nombre as string  private edad as string  sub new(clave as string, nombre as string, edad as integer)  me.clave=clave  me.nombre=nombre  me.edad=edad  end sub  public sub mensaje  console.write("Hola ")  console.writeline(traeDatos())  end sub  public function traeDatos()  return "soy "+me.nombre+" mi clave es:"+me.clave+" y mi edad es:"+me.edad  end function  end class  module prueba  sub main  dim p as new persona("01","juan",23)  p.mensaje  console.readkey()  end sub  end module |

Su salida es: ***hola soy juan mi clave es:01 y mi edad es:23***

Todo bien hasta aquí, ahora suponga que crea una clase alumno que hereda de persona y agrega el atributo carrera:

|  |
| --- |
| Archivo:prueba.vb |
| class persona  private clave as string  private nombre as string  private edad as string  sub new(clave as string, nombre as string, edad as integer)  me.clave=clave  me.nombre=nombre  me.edad=edad  end sub  public sub mensaje  console.write("Hola ")  console.writeline(traeDatos())  end sub  public function traeDatos()  return "soy "+me.nombre+" mi clave es:"+me.clave+" y mi edad es:"+me.edad  end function  end class  class alumno  inherits persona  private carrera as string  sub new(clave as string, nombre as string, edad as integer, carrera as string)  mybase.new(clave,nombre,edad)  me.carrera=carrera  end sub    end class  module prueba  sub main  dim p as new alumno("01","juan",23,"ICO")  p.mensaje  console.readkey()  end sub  end module |

Su salida es: ***hola soy juan mi clave es:01 y mi edad es:23***

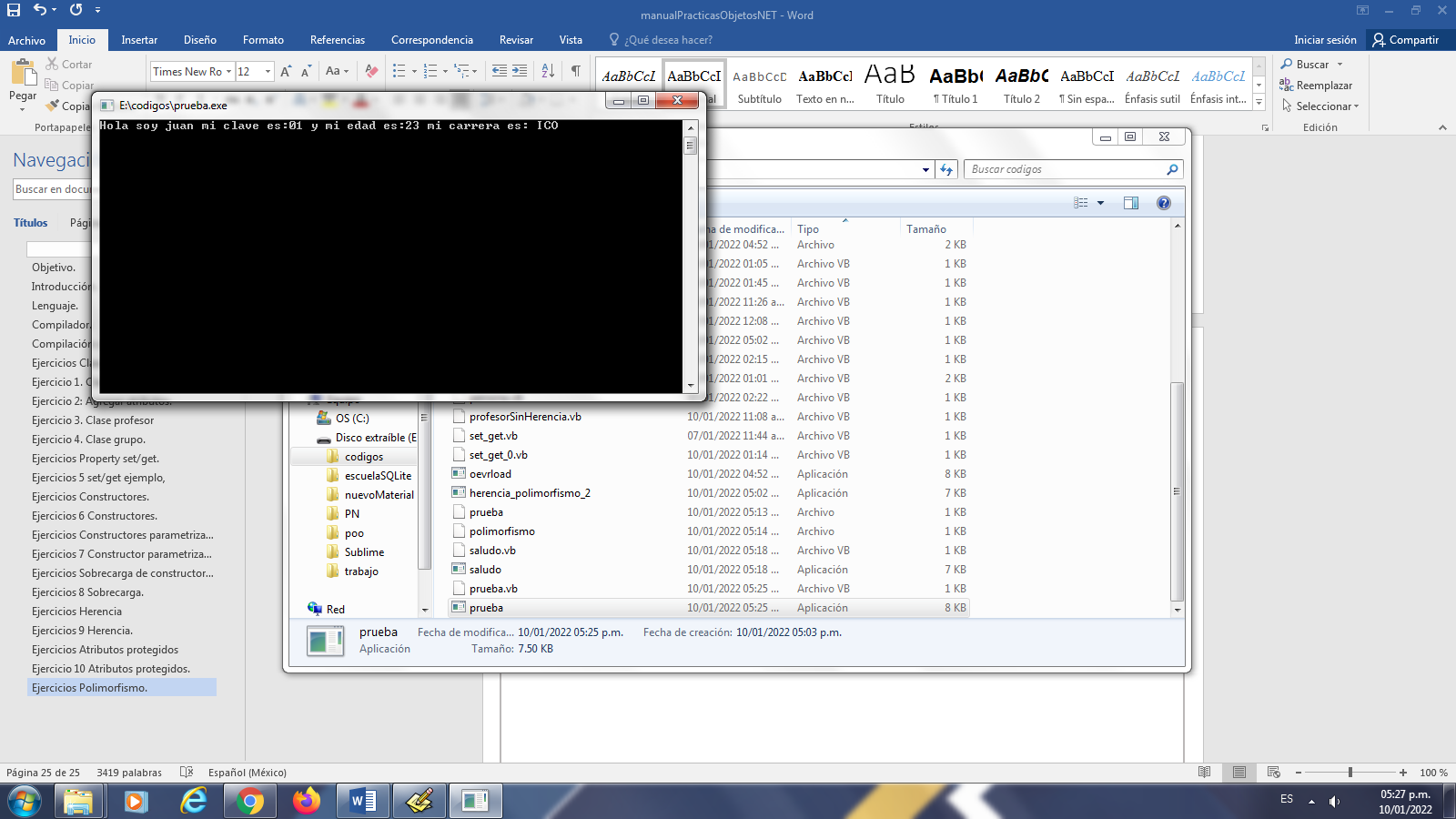
No aparece la carrera, esto es lógico porque no existe un método que escriba los datos del alumno, agreguémoslo:

|  |
| --- |
| Archivo: overrides.vb |
| class persona  private clave as string  private nombre as string  private edad as string  sub new(clave as string, nombre as string, edad as integer)  me.clave=clave  me.nombre=nombre  me.edad=edad  end sub  public sub mensaje  console.write("Hola ")  console.writeline(traeDatos())  end sub  overridable function traeDatos()  return "soy "+me.nombre+" mi clave es:"+me.clave+" y mi edad es:"+me.edad  end function  end class  class alumno  inherits persona  private carrera as string  sub new(clave as string, nombre as string, edad as integer, carrera as string)  mybase.new(clave,nombre,edad)  me.carrera=carrera  end sub  overrides function traeDatos()  return mybase.traeDatos()+" mi carrera es: "+ me.carrera  end function  end class  module prueba  sub main  dim p as new alumno("01","juan",23,"ICO")  p.mensaje  console.readkey()  end sub  end module |

Aquí hicimos algunas cosas:

1. Agregamos un método para escribir los datos del alumno en la clase alumno.
2. Agregamos la palabra reservaba overridable en la clase base y overrides en la clase derivada.

Al probarlo la salida es:



Overridable en la clase base, indica que el método puede ser reescrito,

Y overrides indica que un método está sobrescribiendo uno heredado, de esta forma al tener una clase derivada (alumno en este caso) la clase derivada sobreescribe el método traeDatos de la clase base, por tanto al tener declarada una instancia ***p*** de la clase alumno, este objeto tiene el método reescrito y no el de la clase base, por tanto para este objeto llamar al método traeDatos() siempre hará referencia al reescrito, pero el de la clase base no desaparece, puede seguir siendo invocado con la sentencia mybase.traeDatos().

En C#, solo cambia que en lugar de usar overridable y overrides, se usa virtual y override:

|  |
| --- |
| Archivo:polimorfismo,cs |
| namespace basicos{  using System;  class persona{  private string Nombre;  protected int Edad;  public persona(string n,int e){  Nombre=n;  Edad=e;  }  public string nombre{  set{Nombre=value;}  get{return Nombre;}  }  public int edad{  set{Edad=value;}  get{return Edad;}  }  public virtual void muestra(){  Console.WriteLine("Nombre es: "+Nombre+" Edad: "+Edad);  }  public void cumpleanos(){  Edad++;  muestra();  }    }  class alumno:persona{  private string Carrera;  public alumno(string n, int e, string c):base(n,e){  Carrera=c;  }  public string carrera{  set{Carrera=value;}  get{return Carrera;}  }  public override void muestra(){  base.muestra();  Console.WriteLine(" Carrera "+Carrera);  }    }  public class holaMundo{  public static void Main(string[] args){  alumno p=new alumno("Andrea",18,"Biomedicina");  Console.WriteLine("Prueba de clases");  p.muestra();  Console.WriteLine("Ahora cumpleaños");  p.cumpleanos();  Console.WriteLine("Ahora SI Aparece la carrera");    }  }  } |