

MANUAL DE APRENDIZAJE DE XNA

I. Introducción

El propósito de este material es proveer el conocimiento necesario para desarrollar juegos en XNA. Pero primero debemos aclarar algunos conceptos y aplicaciones que utilizaremos en el material.

XNA en un marco de trabajo que ha sido enfocado en los videojuegos, desarrollado por Microsoft, el cual esta diseñado para trabajar en conjunto con la versión gratuita de su entorno de programación, Visual Estudio¹.

Visual Estudio, es un entorno de programación de última generación, diseñado para construir aplicaciones y software de computadora; XNA por su parte permite utilizar este entorno específicamente para el desarrollo de juegos de video para computadores y la consola de videojuegos de Microsoft, Xbox 360.

En resumen podemos pensar en XNA como un plug-in de Visual Estudio Express que nos da la funcionalidad necesaria para hacer juegos.

2. Requerimientos

Los requerimientos para instalar XNA en el computador son los siguientes:

- Es primordial instalar Visual Studio Express 2005:
<http://msdn.microsoft.com/vstudio/express/visualcsharp/>
- Necesitará XNA Game Studio Express:
<http://msdn.microsoft.com/directx/xna/gse/>
- Es Opcional SDK (librerías de desarrollo) de Microsoft Direct X, las cuales se usarían para el desarrollo de contenidos de audio e imágenes². Sin embargo en nuestros cursos las usaremos directamente.

Para un funcionamiento óptimo de estas aplicaciones los requerimientos de su computador debe tener como mínimo:

- Windows XP actualizado con la última versión de Service Pack³.
- Memoria RAM al menos de 512 MB.

¹ Visual Estudio Express (Usando el lenguaje de programación C#).

² Microsoft Direct X SDK: <http://msdn.microsoft.com/directx/sdk/>

³ La compatibilidad con Windows Vista al momento en que se redactó este tutorial era limitada por lo que no se garantiza el funcionamiento completo de los tutoriales en ese sistema operativo.

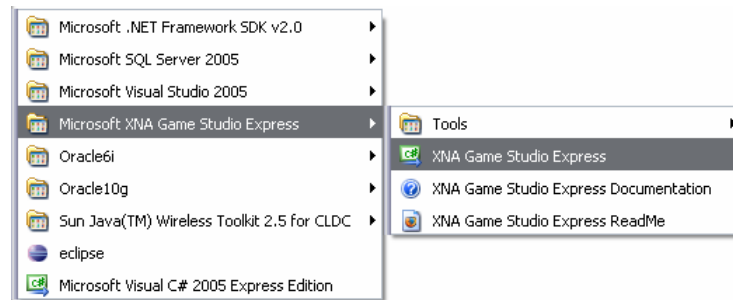
- Una tarjeta de video año 2003 o superior. Soporta las tarjetas NVIDIA GeForce FX series y las ATI Radeon 9500 series en adelante. Si desconoce el tipo o año de la tarjeta, debe mirar en el manual de la misma sea capaz de “soportar DirectX 9.0” o versión superior.
- Requiere un espacio aproximado de 2 GB libres en el disco duro.

De tu parte, aunque no es una camisa de fuerza, pero es muy ventajoso, comenzar este curso con:

- Conocimientos básicos sobre programación en C# o Java.
Programación orientada a objetos.
Estructuras condicionales y cíclicas.

3. Entorno XNA⁴

Primero Inicialicemos XNA



*Figura 1
Acceso directo a XNA*

Para comenzar, inicialicemos XNA Game Studio Express y conozcamos el entorno de la aplicación (figura 2). En el área central, organizada en tabs o páginas, observará una página con el nombre de **Start Page** donde aparecen 4 cuadros. El primero **Recent Projects** muestra los últimos proyectos trabajados y dos vínculos para abrir y crear proyectos. El segundo **Getting Started** es un cuadro de ayuda para utilizar Visual C# Express y conectarse a la comunidad C#. El tercer cuadro **Visual C# Express Headlines** se presenta comentarios del diseñador web de Microsoft. El último cuadro y el más grande presenta los títulos de las publicaciones realizadas por Microsoft relacionadas con C# Express.

⁴ Entiéndase que XNA no es un programa aparte de Visual C# Express, por lo que sería indistinto hablar de XNA o Visual C# Express una vez que hallamos instalado el XNA.

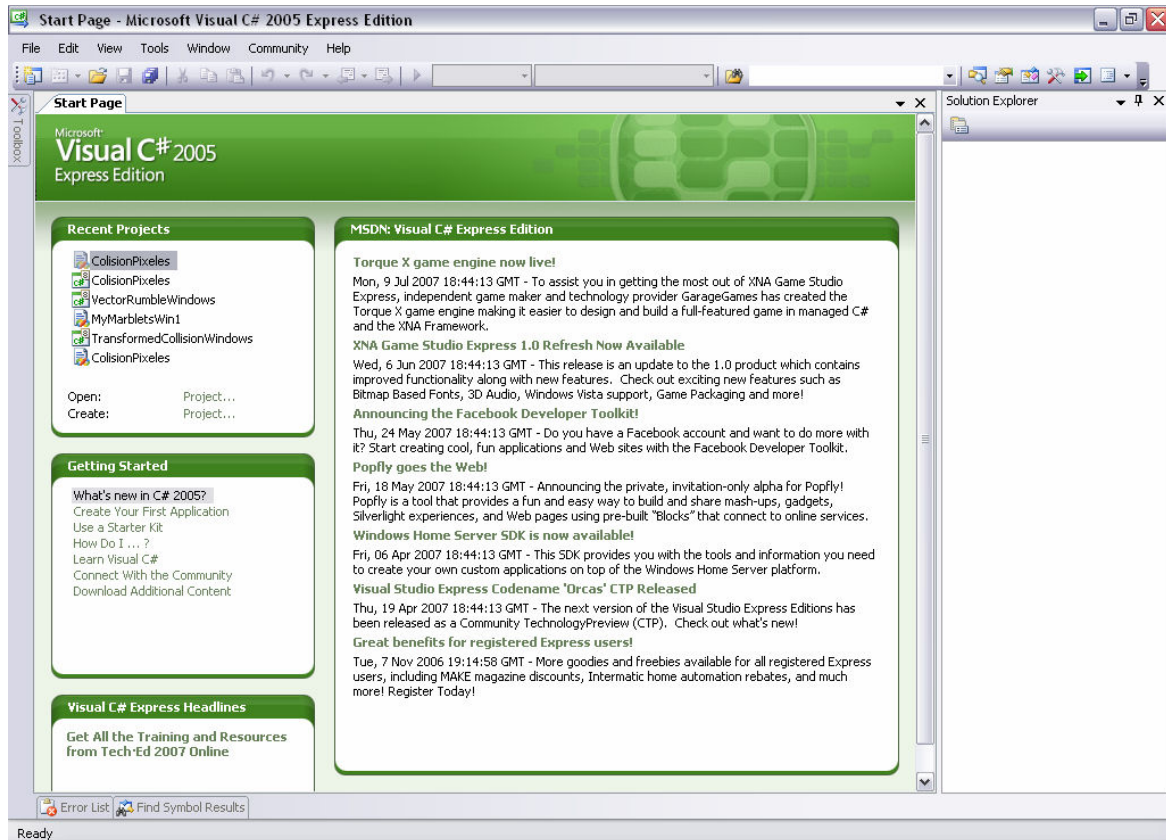


Figura 2
Entorno de trabajo XNA

En el área derecha encuentra el **Solution Explorer** o Explorador de contenido de proyectos y en el área izquierda encontrará una barra donde se encuentra el **Toolbox** la caja de herramientas; ahora no observamos nada en ellos, pero podremos ver su funcionalidad más adelante cuando trabajemos sobre un proyecto.

Así que procedamos a construir un proyecto. Vamos a la barra de menú, ingresamos en **File, New Project** (figura 3).

Un proyecto es la organización más grande que podremos tener en el entorno de programación de Visual estudio. En un proyecto, usualmente definiremos nuestra aplicación (para el caso nuestro un juego) y todos los contenidos necesarios para su funcionamiento (código fuente, imágenes, sonidos, videos, etc).

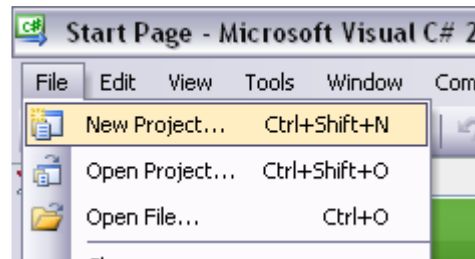


Figura 3
Crear un nuevo proyecto XNA

Al iniciar un nuevo proyecto podemos ver la siguiente ventana (figura 4).

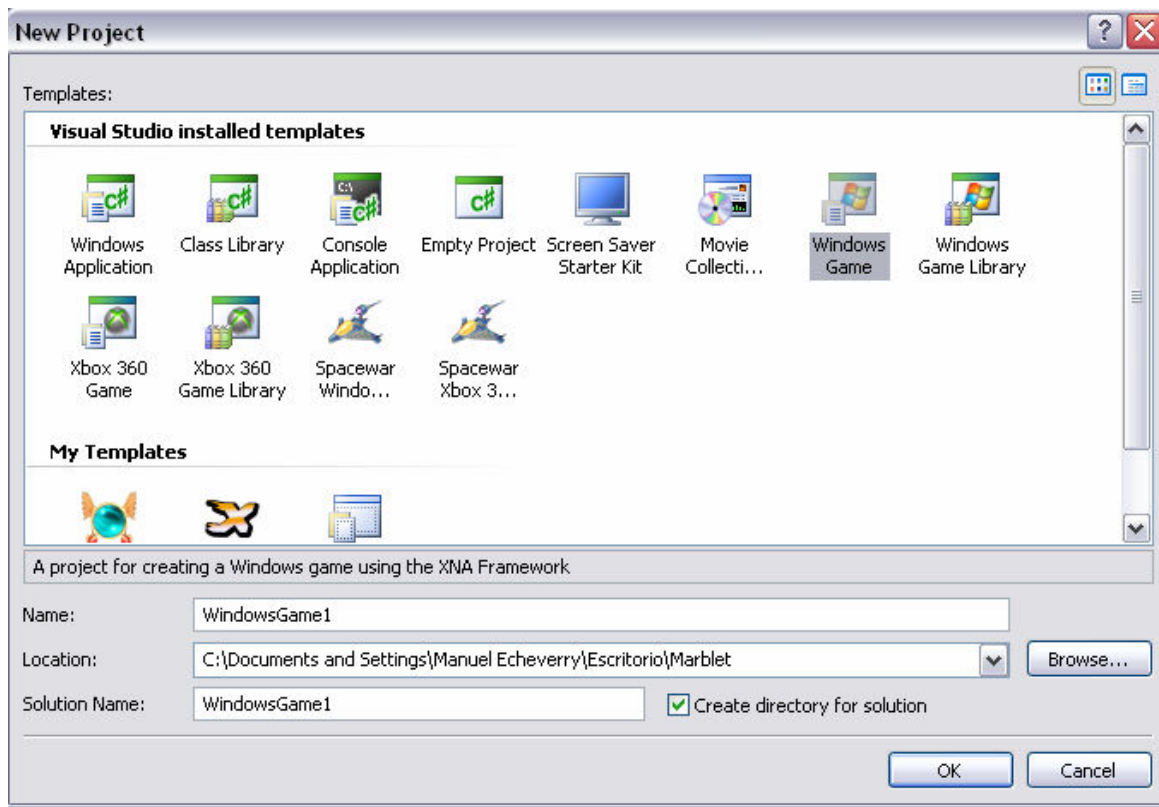


Figura 4
Opciones de nuevo proyecto XNA

Esta ventana contiene todos los tipos de proyectos que se pueden realizar con XNA y visual estudio. Para nuestro caso estamos interesados en realizar un proyecto del tipo de videojuegos **Windows Game (XNA)**, esto significa que el entorno de trabajo se va a preparar para que podamos desarrollar un videojuego cargando las librerías apropiadas. Cabe anotar que el título “Windows Game” no se limita solo a juegos para el sistema operativo Windows; mas adelante veremos

que un juego Windows Game puede ser usado también por ejemplo en la Xbox 360 o en una computadora de bolsillo (PDA)⁵ si se realizan los ajustes necesarios. Sin embargo si te fijas, ya hay también en la lista, un proyecto de tipo Xbox 360 Game, por lo que si esa consola es nuestra meta para el juego a desarrollar, deberíamos usar esa plantilla de una ves.

Selecciona el icono **Windows Game** y colocale un nombre al proyecto, en este caso lo nombraremos “WindowsGame1” el nombre por defecto, pero tú puedes usar el nombre que quieras. Observará que en el área central se crea una nueva página con el nombre de Game1.cs (figura 5).

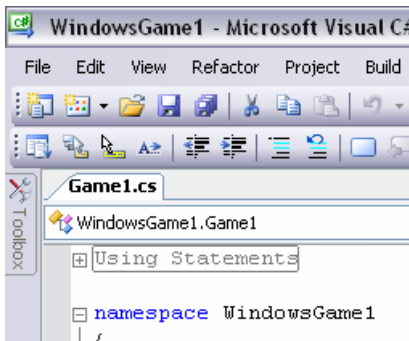


Figura 5
Proyecto Windows Game

Si ejecutamos nuestro proyecto (**F5**) o en la barra de Menú, **debug - start debuggin**, aparecerá una ventana con el nombre del proyecto y un fondo azul (figura 6). Allí visualizará el video juego, y validará su funcionamiento a medida que lo vamos desarrollando. Podemos cerrar y continuar.

Ejecutar un proyecto, equivale a iniciar el juego como si ya estuviera listo. La ventaja de ejecutar un proyecto en visual estudio (mientras lo programamos), radica en que podemos probar los resultados del código que vamos escribiendo. Para ejecutar un proyecto vasta con presionar la tecla (**F5**) o dar clic en el botón con forma de play que hay en la barra de tareas. Al presionar este botón XNA revisa las sentencias de código escritas para corroborar si están bien escritas, y luego si las ejecuta. En caso de encontrar errores avisara y abortara la ejecución del programa.

En caso de encontrar errores en el código, se despliega una lista con los mismos. La experiencia con Visual Estudio, te dará la práctica que necesitas para saber que hacer con cada mensaje de error, de momento, si sigues todos los pasos al pie de la letra, no deberías recibir errores por el código que escribas.

⁵ Los aplicativos para PDA tienen por diversas razones, limitaciones de espacio y velocidad de procesador; por lo que no siempre será posible importar un Windows game a una PDA.

Sin embargo ten presente que hay otro tipo de errores que te pueden salir cuando ejecutes el juego, relacionados con el ambiente de trabajo en el que te encuentres, por ejemplo te podrían salir errores por razones como las siguientes:

- La tarjeta de video no cumple los requisitos mínimos para XNA.
- No tienes permisos de administrador en la maquina en la que trabajas y esto le impide a XNA por ejemplo leer/escribir en algún disco.

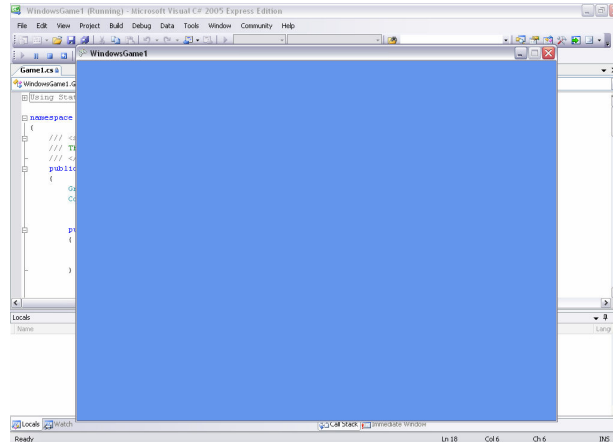


Figura 6

Ejecutar un proyecto Windows Game en blanco

Vamos a mirar ahora el entorno de trabajo de nuestro proyecto. En el Explorador de Contenido (figura 7) se nos muestran los elementos de nuestro proyecto “WindowsGame1”, el diagrama en forma de carpetas nos resulta natural. En cada carpeta que se crea, podremos almacenar los activos (ya sea código, imágenes, sonido o video) que vamos a utilizar en nuestro juego. Por defecto, XNA crea una plantilla de juego en blanco que nos servirá de guía para programar nuestro juego (que es la que estamos viendo ahora). Estos contenidos básicos pueden ser revisados con este explorador.

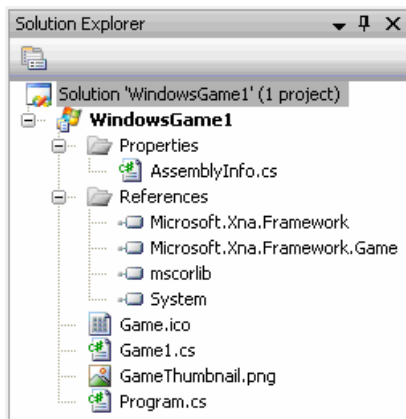


Figura 7

Explorador de contenido

4. Código generado

Ahora comenzamos a relacionarnos con el código del proyecto. Ve al Explorador de Contenido y dale doble clic sobre el archivo Game1.cs. Aparecerá una nueva página con el código del proyecto. El **namespace** WindowsGame1, es el espacio de programación de nuestro proyecto y guardará todas las clases que desarrollemos.

Antes de continuar vamos a hacer algunas precisiones sobre los términos que vamos a utilizar de aquí en adelante.

C# es un lenguaje de programación de alto nivel, lo que significa que fue diseñado para facilitarle el desarrollo a los programadores; esto es posible debido a que implementa la filosofía de “Programación Orientada a Objetos o POO”. La programación orientada a objetos, le permite al programador entre otras ventajas, realizar una abstracción de la realidad por medio de unas estructuras que organizan el código y le permiten usarlo de manera más eficiente y fácil que los lenguajes no orientados a objetos.

Las estructuras básicas de la orientación a objetos en C# son:

- El namespace.
- La Clase
- El método

Los Namespaces: Podemos hacer una analogía de un namespace con una carpeta de archivos, de esta manera nuestro código se puede separar en carpetas para conservar el orden. De hecho, cada namespace efectivamente corresponderá a una carpeta de Windows en nuestros archivos del proyecto; no obstante esto no es estrictamente necesario pues uno podría darles nombres diferentes a los de las carpetas, pero por motivos de orden y facilidad se recomienda mantener los nombres iguales.

Las Clases: Es la siguiente estructura en nuestra organización. Una clase se puede asociar como su nombre lo indica, con una categoría; por ejemplo una clase podrían ser los “Carros” y otra las “motos” y todo el código relacionado con los carros iría en la clase de los carros así como el de las motos.

Siguiendo con nuestra analogía, si los namespaces son las carpetas que contienen nuestros archivos del proyecto, las clases son efectivamente, los archivos que se colocan dentro de cada carpeta. En otras palabras los namespaces pueden contener muchas clases y las clases no son más que archivos que tienen en su interior el código correspondiente a esa clase.

Los métodos: Finalmente, tenemos los métodos, los cuales representan todos aquellos servicios que una clase puede ofrecernos; para el ejemplo de la clase carro, podríamos tener métodos como “acelerar”, “frenar”, “pitar” y cualquier otro servicio que pueda prestarnos el carro.

En otras palabras los métodos se corresponden a fragmentos código escrito que hay en cada clase (una vez mas, por razones de orden, es que el código se divide en métodos).

Estos conocimientos básicos nos serán suficientes por ahora para continuar explorando XNA. Posteriormente iremos precisando más las definiciones y los conceptos de programación.

Regresando al proyecto en blanco que teníamos abierto, en el explorador de soluciones podemos ver la clase **Program** (figura 11) (note que pertenece al namespace **WindowsGame1**)⁶ Esta clase contiene el método denominado **main** (figura 11) que es el punto de inicio de cualquier proyecto. Sencillo hasta el momento, este método lo que hace es permitir que nuestro juego ejecute todo lo que escribamos en la clase Game1.

La clase **program** que creó automáticamente XNA cuando le dijimos que queríamos hacer un Windows Game, es la clase en la que se ejecutará el juego; es decir, esta clase hará uso del código que se encuentre en todas las demás clases. El método **main**, representa en cualquier tipo de programa (no solo en los juegos) el punto de partida de una aplicación. Es decir que a partir de este método se empiezan a leer las instrucciones, por lo que este es un método de carácter obligatorio y debe respetarse su nombre.

Ampliando nuestras definiciones iniciales, entonces podemos decir que los métodos son agrupaciones de código a las cuales se les ha puesto un nombre para poder ser usadas en muchos lugares diferentes de nuestro programa. Puede identificar un método por que tiene un nombre luego abre y cierra paréntesis y luego abre llaves, posteriormente hay código del método como tal, y finaliza con llaves nuevamente, por ejemplo:

```
miMetodo( ){  
    // Código del método  
}
```

Las demás partes de un método (indicadores de acceso [*que van antes del nombre*], parámetros [*que van dentro de los paréntesis*] y valor de retorno [*que va después del indicador de acceso*]) serán discutidas posteriormente.

⁶ Puedes navegar por el disco duro de tu equipo en la carpeta en la que gravaste el proyecto (mis documentos es la ruta por defecto) para verificar que efectivamente el namespace es una carpeta y la clase program un archivo.

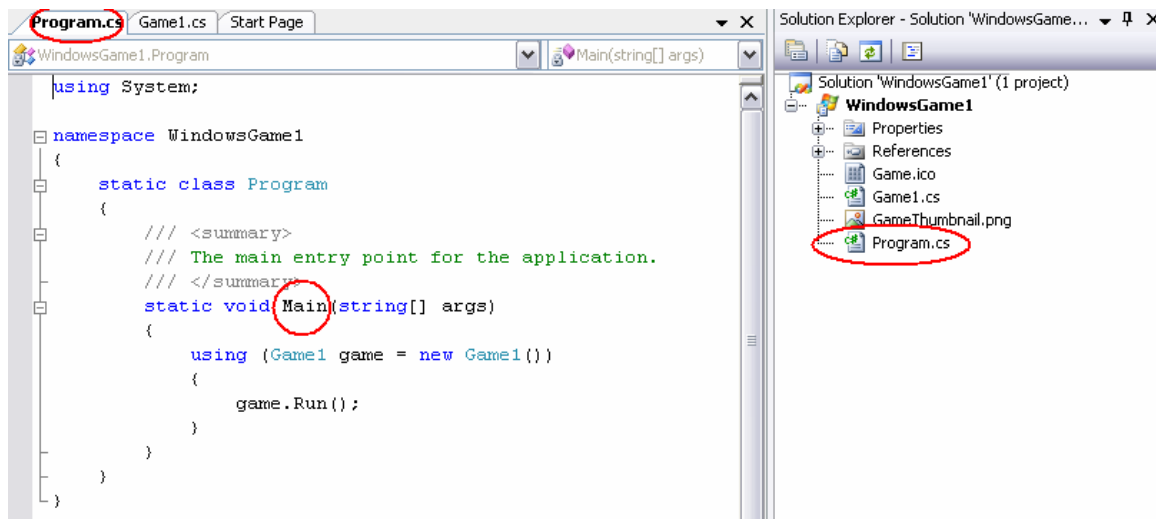


Figura 11
El método main

Si continuamos explorando nuestro entorno de trabajo, notaremos que en la parte superior (figura 12), aparecen dos campos con listas desplegables; la primera despliega las clases con su **namespace** al que pertenecen y la segunda los métodos de la clase seleccionada en el primer campo. El uso de estas listas es una técnica que nos permitirá movilizarnos con mayor facilidad a medida que el proyecto crezca.

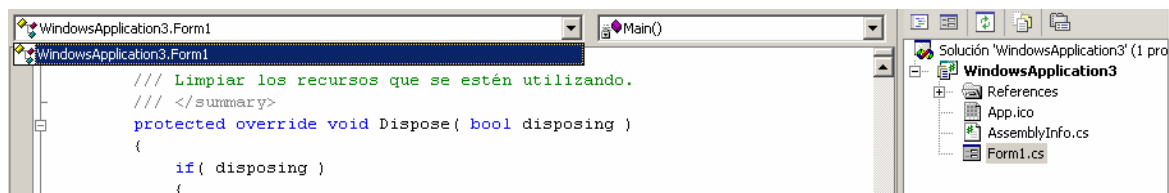


Figura 12
Listas de navegación de clases y métodos

Por ahora puedo asegurar que entendemos que una clase es una agrupación de métodos, así pues, una clase agrupa un conjunto de métodos (servicios) que podremos pedirle a ella que realice. Cabe anotar que una clase también puede contener variables a las cuales denominaremos atributos. Por ejemplo:

```
MiClase{
    // Atributos
```

```
miMetodo( ){  
    // Código del método  
}  
  
OtroMetodo( ){  
    // Código del método  
}  
}
```

Los atributos constituyen características comunes a los objetos de una clase, es decir, para la clase Carro de la que hablamos previamente, podríamos definir atributos como color, matricula, kilometraje, modelo, etc. Es decir que cada carro que construyamos con esa clase tendría que definir valores propios para esos atributos.

El concepto de objeto entonces aparece ante nosotros como la instancia (el carro que construimos) a partir de una clase. Es decir que cada carro que construyamos (instanciemos) con la clase Carro, es un objeto del tipo Carro.

Volvamos nuevamente al Explorador de contenido y dale clic derecho sobre la clase Game1.cs⁷ y selecciona “Ver código” (**View code**).

El código que aparece es el de la clase Game1 que es hija de la clase Microsoft.Xna.Framework.Game es decir hereda la estructura que explicaremos a continuación la cual es necesaria para crear un videojuego en XNA.

Cuando hablamos de que una clase hereda de otra nos referimos aun concepto poderoso de la programación orientada a objetos, el cual esta relacionado con la reutilización de código.

Para nuestro ejemplo de la clase carro, imaginemos que tenemos una clase mas general como Vehiculo que tiene atributos y métodos que cualquier tipo de vehiculo debería tener; por ejemplo atributos como color, velocidad máxima, modelo, tipo de motor, etc y métodos como encender, apagar, acelerar, frenar. De esta manera nuestra clase Carro podría heredar (ser hija de) la clase Vehiculo y de esta manera no tendría que definir ni los atributos ni los métodos de su padre pues por ser hija de el ya los tiene, así pues otras clases que se nos ocurran, como Lancha y Moto podrian heredar tambien de Vehiculo.

La sintaxis de herencia es muy sencilla en C#; basta con colocar el nombre de la clase hija y seguido de dos puntos “:” el nombre de la clase padre.

⁷ La extensión .cs hace referencia al lenguaje de programación utilizado (en nuestro caso C#)

```

MiClase : ClasePadre{

    // Atributos

    metodo( ){
        // Código del método
    }
}
  
```

Retomando nuestro código, la clase **Game1** (uno puede cambiar este nombre a gusto propio) es la clase principal que XNA nos crea automáticamente. En ella se encuentran los métodos necesarios para hacer que un Windows Game se muestre en la pantalla; inicialmente solo coloca el código necesario para crear una ventana vacía de color azul.

La primera parte del código (figura 13) es la implementación de varias librerías que se van a usar en el desarrollo del código para el juego. Las librerías se refieren a código que los creadores de XNA ya han creado y lo han puesto a disposición de nosotros para evitarnos mucho trabajo innecesario; de esta manera ya existen por ejemplo librerías que le indican a la pantalla cuando y que pintar, según los parámetros que nosotros le demos.

Para implementar una librería en XNA se utiliza la palabra clave **“using”** y el nombre de la librería⁸ siguiendo la estructura de namespaces en la que se encuentre”. Esto pone a nuestra disposición, una funcionalidad deseada.

```

using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Components;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Storage;
  
```

Figura 13
Librerías de un Windows Game

Como vemos la clase Game1 tiene un constructor y otros métodos (ver figura 14). El constructor, no es más que otro método, cuya característica principal es que

⁸ Note que la librería no es mas que el nombre de una clase, por lo que implementar una librería es decirle a nuestro programa que puede utilizar los métodos de una clase. Cabe anotar que heredar de una clase es diferente de implementar una clase, pues al implementar una clase, nuestra clase no hace los métodos de la otra clase como propios.

posee el mismo nombre que el de la clase. Como veremos mas adelante, el constructor, tiene la característica de permitir que la clase sea instanciada (es decir permite crear los objetos de esa clase, para el ejemplo de la clase Carro, es el método que no entrega carros nuevos cuando se lo pedimos).

Ahora, una instancia de una clase, es decir un objeto, es una entidad propia e independiente, que puede prestar todos los servicios (métodos) de la clase que instancia.

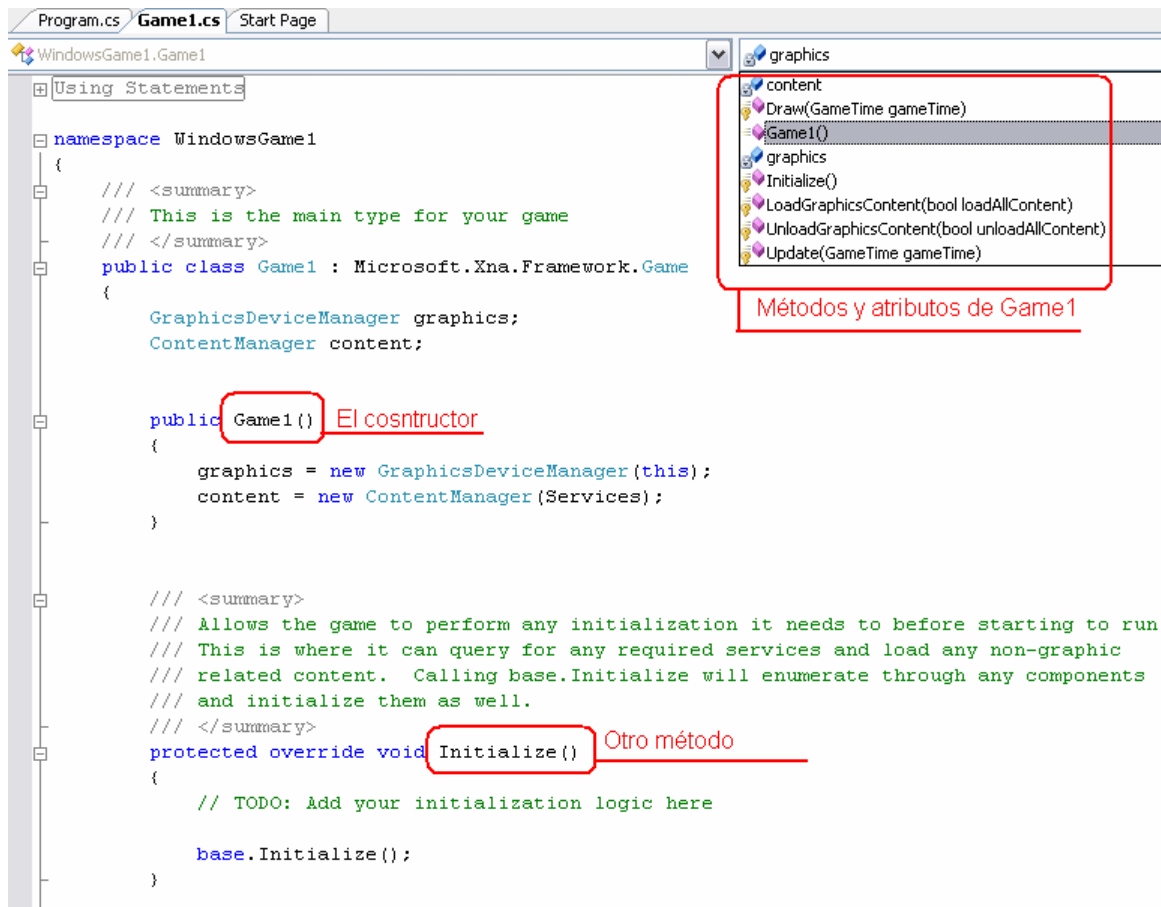


Figura 14
Estructura de la clase Game1

De los métodos que hay en la clase `Game1` vamos a ir hablando poco a poco en lo corrido de estos tutoriales, sin embargo vamos a introducir los 2 más importantes.

El método **Update** es llamado una y otra vez mientras el juego se ejecuta. Aquí sucede toda la acción. Desde él se manejan los movimientos del jugador, entradas del control, etc., actualizando todos los cambios que se realizan en el juego.

El método **Draw** es el corazón y el alma del juego. Su función es colocar todas las imágenes en la pantalla, algunas de las cosas que el hace por nosotros son:

- Se asegura que el dispositivo de gráfico sea válido.
- Limpia el fondo de la pantalla con el color azul que vimos anteriormente.
- Le dice a la tarjeta de video que se va a dibujar una escena.
- Le dice a cada uno de los componentes que heredan el método que dibujen.
- Le dice a la tarjeta de video cuando ha terminado de dibujar la escena.
- Le dice a la tarjeta de video que muestre los resultados.

TEMA COMPLEMENTARIO

La sintaxis del lenguaje C# es otro tema que vamos a explorar sobre la marcha, no obstante es importante que hagamos algunas precisiones antes de empezar a programar.

Palabras reservadas

Como todo lenguaje de programación (así como cualquier lenguaje en general), C# se compone de un conjunto de palabras (denominadas palabras reservadas) que conforman el lenguaje con el que le vamos a decir al computador que queremos que haga. El hecho de que sean reservadas, significa que son exclusivas del lenguaje y no las podremos usar como nombre de nuestras variables o clases. Las palabras reservadas de C# son:

abstract	event	new	struct	do
as	explicit	null	switch	double
base	extern	object	this	else
bool	false	operator	throw	enum
break	finally	out	true	is
byte	fixed	override	try	lock
case	float	params	typeof	long
catch	for	private	uint	namespace
char	foreach	protected	ulong	sizeof
checked	goto	public	unchecked	stackalloc
class	if	readonly	unsafe	static
const	implicit	ref	ushort	string
continue	in	return	using	while
decimal	int	sbyte	virtual	get
default	interface	sealed	volatile	set
delegate	internal	short	void	where

El significado y uso de ellas será estudiado mas adelante, por el momento podríamos reflexionar sobre el hecho de que un lenguaje se componga de tan pocas palabras (a diferencia de un lenguaje humano como el español que tiene millones de palabras). Hemos resaltado en negrilla las palabras reservadas que usaremos mas comúnmente durante el desarrollo de nuestros tutoriales.

Operadores.

C# proporciona un amplio conjunto de operadores, que son símbolos que especifican las operaciones que se deben realizar en una expresión. C# dispone de los operadores aritméticos y lógicos habituales, y de una gran variedad de otros operadores, como se muestra en la siguiente tabla.

Categorías	Operadores
Aritméticos	+ - * / %
Lógicos (booleanos y bit a bit)	& ^ ! ~ && true false
Concatenación de cadenas	+
Incremento y decremento	++ --
Desplazamiento	<< >>
Relacionales	== != < > <= >=
Asignación	= += -
Acceso a miembros	= *= /= %= &= = ^= <<= >>= ??
Indización	.
Condicionales	[]
Creación de objetos	?:
Información de tipos	new
	as is sizeof typeof

El uso de los operadores será estudiado mas adelante, no obstante es importante conocer de la existencia de estos operadores y de las palabras reservadas para que no se nos hagan extrañas en próximas lecciones.

Hemos resaltado con negrilla los operadores que usaremos más comúnmente durante el desarrollo de nuestros tutoriales.

Tipos de datos

Son las construcciones que se an creado en los lenguajes de programación para almacenar valores especificos. Algunos tipos de datos son comunes en todos los lenguajes de programación, vamos a ver los mas comunes de c#.

Numéricos: como su nombre lo indica, representa números.

- **Datos enteros:** permiten guardar números enteros (sin decimales), su palabras reservadas son **int**, **long** y **short** cada una define si rango, y el de la segunda es el doble de grande, puede tomar valores positivos y negativos.

```
int valor = 5454;  
long valor2 = 34432223;
```

- **Datos no enteros:** Permiten guardar datos en formato decimal en diferentes rangos según la palabra clave que se utilice (**double**, **float**,)

```
float valor = 5.53f; // observe que en este caso hay que colocar una f al final  
double valor2 = 8.345f;
```

Boléanos: Representan datos que tienen solo 2 valores posibles.

- **bool:** puede tomar 2 valores, verdadero (true) o falso (false), su palabra reservada es **bool**.

```
bool valor = true;
```

De texto: Almacenan datos en formato no numéricos es decir, cadenas de caracteres.

- **string:** Permite almacenar cadenas de caracteres en formato universal (unicode), su palabra reservada es **string**, las cadenas de texto se encierran entre comillas dobles.

```
string cadena = "Esta es una cadena de texto";
```

- **char:** permite almacenar un único carácter unicode. Su palabra reservada es **char**, y los caracteres se escriben entre comillas sencillas.

```
char letra = 'a';
```

Explorar más.

Si quieres explorar C# mas a fondo antes de embarcarte en el próximo tutorial, puedes leer la guía introductoria de programación en C# oficial disponible en línea. [http://msdn2.microsoft.com/es-es/library/67ef8sbd\(VS.80\).aspx](http://msdn2.microsoft.com/es-es/library/67ef8sbd(VS.80).aspx)

O desde XNA puedes acceder también en el menú **help** - **how do I**. (ver inicialmente solo los 3 primeros apartados.