



Taller de Programación Visual III

Daniel Alejandro Nuño
Ramirez

Presentación

- Daniel Alejandro Nuño Ramirez
- Ingeniero en Computación, Universidad de Guadalajara
- Maestro en Informática Aplicada*, ITESO
- Gerente de Desarrollo de Software, Oracle
- TCS, Intel, Bank of America, Flextronics, Estratel, UdeG
- Correr, Cine, Series, Música
- Email: daniel_nuno@hotmail.com

Contenido

Unidad I, Introducción a Microsoft Visual Studio .NET

Unidad II, Introducción a Microsoft Visual C#

Unidad III, C# Avanzado

Unidad IV, Exploración de la biblioteca C#

Unidad V, Programación multiproceso

Unidad VI, Creación de aplicaciones Windows

Unidad VII, El modelo de acceso a datos ADO.NET



Unidad I, Introducción Microsoft Visual Studio .NET





Unidad I, Introducción Microsoft Visual studio .NET

1. El entorno de desarrollo de Visual Studio .NET
2. Tipos de proyectos en Visual Studio .NET
3. Elementos esenciales de la plataforma .NET
4. Componentes y objetos de la plataforma .NET
5. Comprensión del modelo de desarrollo .NET
6. La estructura básica de .NET Framework.

Unidad I, Introducción Microsoft Visual Studio .NET

- C# vs .NET
 - C# es un lenguaje de programación
 - .NET es un plataforma para construir aplicaciones con Windows
 - No se limita a C#, se puede usar F# o VB.NET
- .NET se conforma por
 - CLR (Common Language Runtime)
 - Class Library

Unidad I, Introducción Microsoft Visual Studio .NET

- Visual Studio (<https://visualstudio.microsoft.com/>)
 - IDE (Integrated Development Environment)
 - Editions (Tamaño, Funcionalidad, Costo)
 - Ultimate (\$)
 - Premium (\$)
 - Professional (\$, equipos pequeños)
 - Test Professional (\$)
 - Express (libre, Web, C#, VB, etc, no se puede utilizar extensiones)
- ➡ Community (libre, 2014, estudiantes, individuos, algunas restricciones)

Visual Studio Enterprise 2017

Visual Studio Professional 2017

Visual Studio Community 2017

Unidad I, Introducción Microsoft Visual Studio .NET

- Visual Studio Community (libre, 2014, estudiantes, individuos, algunas restricciones)
 - Web, Windows, Desktop y aplicaciones móviles
- Profesional con MSDN
 - Incluye todas las características para crear aplicaciones de todo tipo con todos los lenguajes de programación disponibles en la plataforma .NET (VB, C#, F#, TypeScript, C++)
- Enterprise con MSDN
 - Antes conocido como Ultimate, incluye las características centrales y herramientas avanzadas para crear aplicaciones y evitar “no se puede reproducir”.

Unidad I, Introducción Microsoft Visual Studio .NET

- Test Professional
 - Herramienta periférica desarrollada para “Testers”
- Visual Studio Core
 - Gratuita, Open Source, orientada para desarrollos Web sobre sistema operativos: Windows, Mac y Linux
 - <https://code.visualstudio.com/>
- Comparativo
 - <https://visualstudio.microsoft.com/vs/compare/>

Unidad I, Introducción Microsoft Visual Studio .NET

- Visual Studio Community (libre, 2014, estudiantes, individuos, algunas restricciones)
 - Web, Windows, Desktop y aplicaciones móviles
 - Estudiantes, equipos pequeños de desarrollo, contribuyentes al open source.
 - No incluye: SharePoint, Office, LightSwitch, y Cloud Business Apps.
 - <https://visualstudio.microsoft.com/es/vs/community/>

Unidad I, Introducción Microsoft Visual Studio .NET

- MSDN
 - Suscripción incluida con la versión Visual Studio Enterprise (\$150 USD) o Profesional (\$50 USD)
 - Permite acceso a TFS o Visual Studio online (VSO), así como también, acceso a: SharePoint, Exchange, Office, Dynamics, BizTalk, LightSwitch, y Cloud Business Apps.
 - <https://www.visualstudio.com/products/visual-studio-with-msdn-overview-vs>

Unidad I, Introducción Microsoft Visual Studio .NET

- TFS
 - Application Lifecycle Management (ALM) Environment
 - Gestionar y dar seguimiento a trabajo
 - Dos versiones:
 - On-premise-hosted TFS
 - Version online, llamada Visual Studio Online (VSO)
 - <https://www.visualstudio.com/en-us/products/what-is-visual-studio-online-vs.aspx>

Unidad I, Introducción Microsoft Visual Studio .NET

- Lenguajes de programación soportados en la plataforma .NET
 - C#, rapid application development,
 - Visual Basic .NET (VB.NET), productividad
 - C++, C background, Active Template Library (ATL), Microsoft Foundation Class (MFC) libraries, C Runtime (CRT) Library.
 - Visual F#, matemáticas, científico, ingeniería, y análisis (Machine Learning)
 - TypeScript, respuesta a JavaScript.

Unidad I, Introducción Microsoft Visual Studio .NET

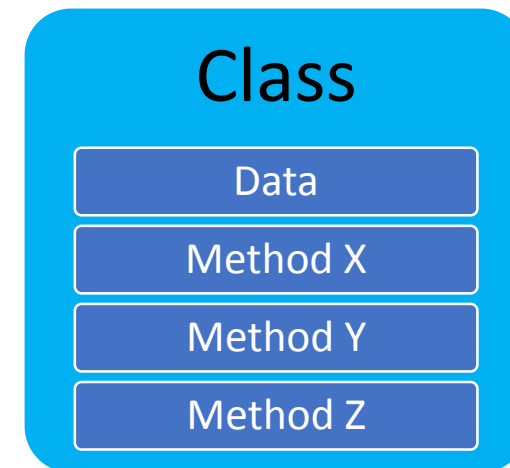
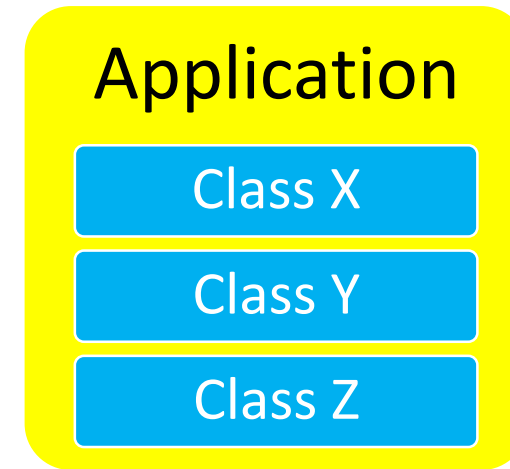
- Actividad en clase:
 - Instalación de Microsoft Visual Studio

Unidad I, Introducción Microsoft Visual Studio .NET

- CLR (Common Language Runtime)
 - Misma idea que la Java Virtual Machine (JVM)
 - Intermediate Language Code (IL Code)
 - CLR es una aplicación que traduce de IL código a código Nativo
 - Just In Time Compilation (JIT)

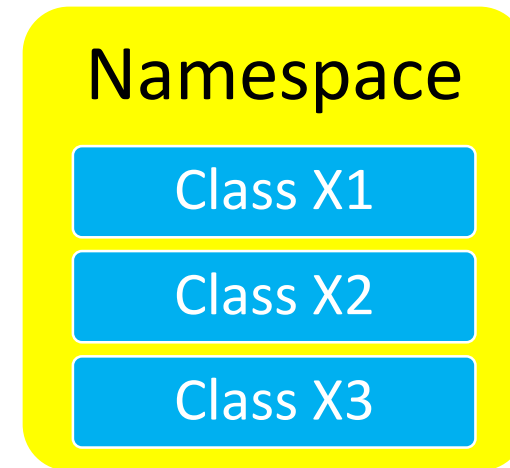
Unidad I, Introducción Microsoft Visual Studio .NET

- Arquitectura de las aplicaciones .NET
- Aplicaciones se forman a base de clases y su interacción mediante métodos



Unidad I, Introducción Microsoft Visual Studio .NET

- Namespaces
 - Colección de clases relacionadas
 - Para acceder a una clase de un namespace, se utiliza la siguiente nomenclatura:
 - `NamespaceName.ClassName.Method`
 - `using (Directive)`
 - Algunos Namespaces:
 - Databases
 - Graphics
 - Security



Unidad I, Introducción Microsoft Visual Studio .NET

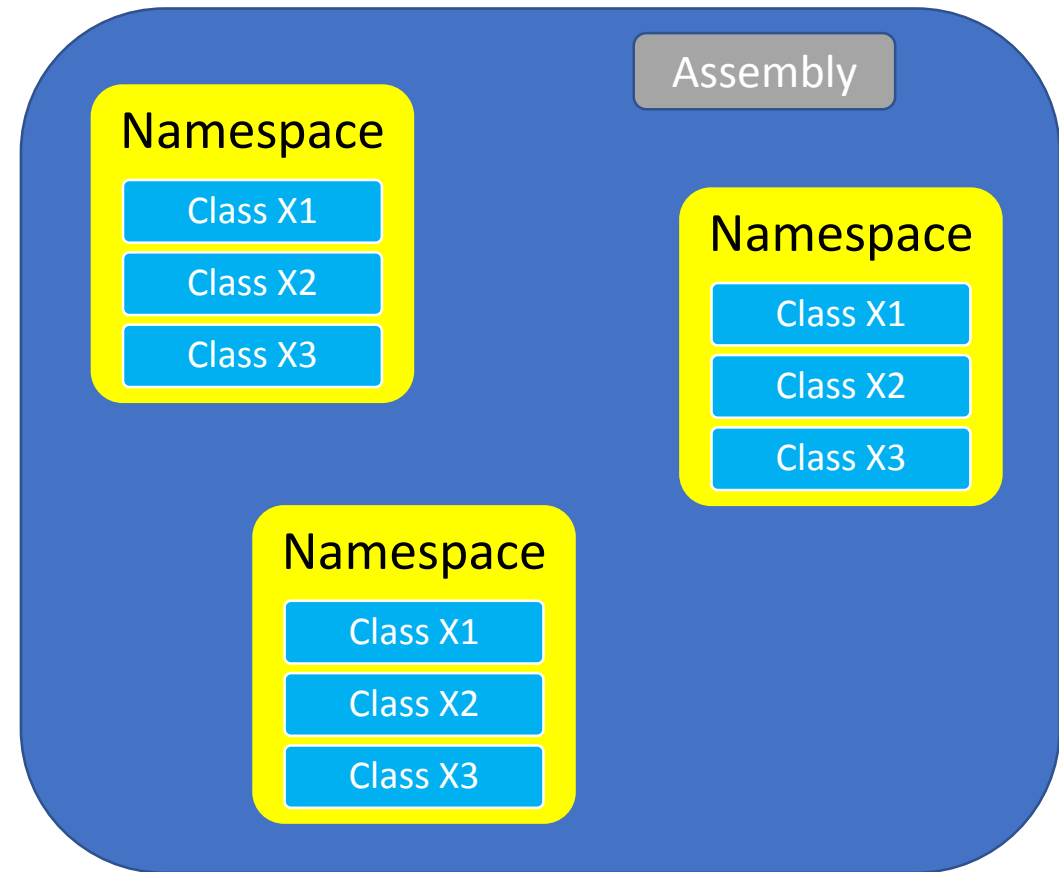
- El namespace global, es el namespace raíz (root)
- global::System se refiere al namespace de la plataforma .NET (System)

Ejemplo:

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

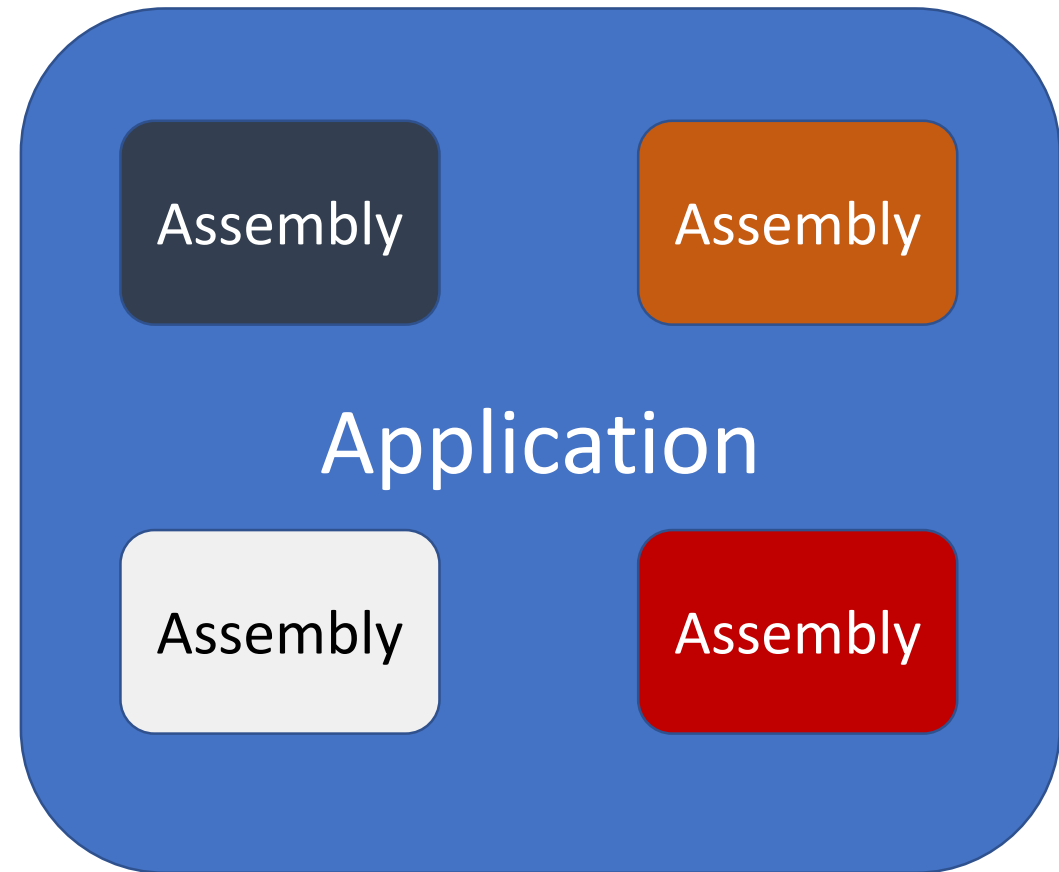
Unidad I, Introducción Microsoft Visual Studio .NET

- Assembly (DLL o EXE)
 - Contenedor de namespaces relacionados
 - Puede ser en formato
 - DLL (Dynamic-link Library)
 - EXE (Ejecutable)




Unidad I, Introducción Microsoft Visual Studio .NET

- Application
 - Assembly
 - Namespace
 - Class
 - Methods
 - Assembly
 - Namespace
 - Class
 - Methods





Unidad II, Introducción a Microsoft Visual C#



Unidad II, Introducción a Microsoft Visual C#

- 2.1 Breve descripción del C#.
- 2.2 Tipos de datos básicos.
- 2.3 La biblioteca de clases.
- 2.4 Operadores.
- 2.5 Instrucciones de control de programa.
- 2.6 Clases, objetos y métodos en C#.
- 2.7 Arreglos y Matrices, Cadenas.
- 2.8 Sobrecarga de operadores.
- 2.9 Indizadores y propiedades.
- 2.10 Herencia en C#.
- 2.11 Interfaces, estructuras y enumeraciones.
- 2.12 Colecciones



Unidad II, Introducción a Microsoft Visual C#

- 2.1 Breve descripción del C#
- C# historia
 - C# se pronuncia “C-Sharp”
 - Lenguaje de programación orientado a objetos
 - Distribuido por Microsoft y se ejecuta sobre la plataforma .NET
 - Anders Hejlsberg es su fundador
 - Basado en C++ y Java
 - Se introdujo en 2002

Unidad II, Introducción a Microsoft Visual C#

2.1 Breve descripción del C#

- El porque estudiar C#
 - Combina las mejores características de otros lenguajes de programación ya existentes tales como: C++, Pascal, Java, Visual Basic
 - Sencillo
 - Lenguaje de programación Moderno
 - Orientado a objetos
 - Seguro
 - Tiene una basta biblioteca
 - Rápida ejecución

Unidad II, Introducción a Microsoft Visual C#

2.1 Breve descripción del C#

- C# se ha convertido en un standard, lo que permite que se puedan hacer otras implementaciones además de Microsoft C#.
 - <http://www.ecma-international.org>
- C# se puede utilizar además en plataformas no Windows a través de Mono Project y .NET core que son manejadas por la fundación .NET
 - <http://www.dotnetfoundation.org/>

Unidad II, Introducción a Microsoft Visual C#

- Algunas capacidades incluidas en su biblioteca:
 - Bases de datos
 - Debugging
 - Web Apps
 - Graphics
 - Input/Output
 - Networking
 - Permissions
- Continuación:
 - Mobile
 - Procesamiento de “Strings”
 - Multithreading
 - File Processing
 - Security
 - Graphical User Interface
 - Data Structures

Unidad II, Introducción a Microsoft Visual C#

C# Version History		
Version	Features	Year of release
C# 1.0	Basic Features	2002
C# 2.0	Generics, Partial types, Anonymous methods, Nullable types, Static classes	2005
C# 3.0	Var, LINQ, Lambda expression, Auto-implemented properties, Anonymous types, Extension methods	2007
C# 4.0	Dynamic binding, Named and Optional arguments	2010
C# 5.0	Asynchronous methods, Caller info attributes	2012
C# 6.0	Auto-property initializers, Null-propagating operator, Exception filters, Using static members, ...	2015

Unidad II, Introducción a Microsoft Visual C#

- Referencias:

C# Reference

- <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/index>

C# Guide

- <https://docs.microsoft.com/en-us/dotnet/csharp/index>

C# Programming Guide

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/index>

Unidad II, Introducción a Microsoft Visual C#

2.2 Tipos de datos básicos

- Primitive Types and Expressions
 - Variables y Constantes
 - Conversión entre tipos

Unidad II, Introducción a Microsoft Visual C#

	C# Type	.NET Type	Bytes	Range
Integral Numbers	byte	Byte	1	0 to 255
	short	Int16	2	-32,768 to 32,767
	int	Int32	4	-2.1B to 2.1B
	long	Int64	8	...
Real Numbers	float	Single	4	-3.4×10^{38} to 3.4×10^{38}
	double	Double	8	...
	decimal	Decimal	16	-7.9×10^{28} to 7.9×10^{28}
Character	char	Char	2	Unicode Characters
Boolean	bool	Boolean	1	True / False

Unidad II, Introducción a Microsoft Visual C#

- Variables

- Variable, localidad de memoria usada para almacenar un valor que puede ser alterado en el tiempo.
- Tipos:
 - Decimal
 - Boolean
 - Integral: int, char, byte, short, long
 - Floating point: float, double
 - Nullable

- Ejemplo:

```
int number;  
int Number = 1;  
double d;  
float f;  
char ch;
```

Unidad II, Introducción a Microsoft Visual C#

- Constantes

- Valor constante asignado a un identificador.
- No cambia en el futuro.
- Declarado con la palabra reservada “const”.

- Ejemplo:

```
const float Pi = 3.14f;
```


Unidad II, Introducción a Microsoft Visual C#

- Consideraciones
 - En C#, no se hace una verificación de “overflowing”;
 - Al declarar un número flotante, se debe especificar que el formato del valor es flotante también, de lo contrario se trataría de usar “double”.

Unidad II, Introducción a Microsoft Visual C#

- Conversiones

- Conversión Implícita:

- `byte b = 1;`
 - `int i = b;`
 - `float f = i;`
 - `byte c = i;`

- Cont.

- Conversión Explícita (casting)

- `byte c = (byte) i;`
 - `int j = (int) f;`

Unidad II, Introducción a Microsoft Visual C#

- Conversiones

- Conversión entre no compatibles:
 - `string s = "1";`
 - `int i = (int) s; // won't compile`
- `int i = Convert.ToInt32(s);`
- `int j = int.Parse(s);`

- Cont.

- `Byte()`
- `ToInt16()`
- `ToInt32()`
- `ToInt64()`

UNIDAD I, introducción a la programación visual

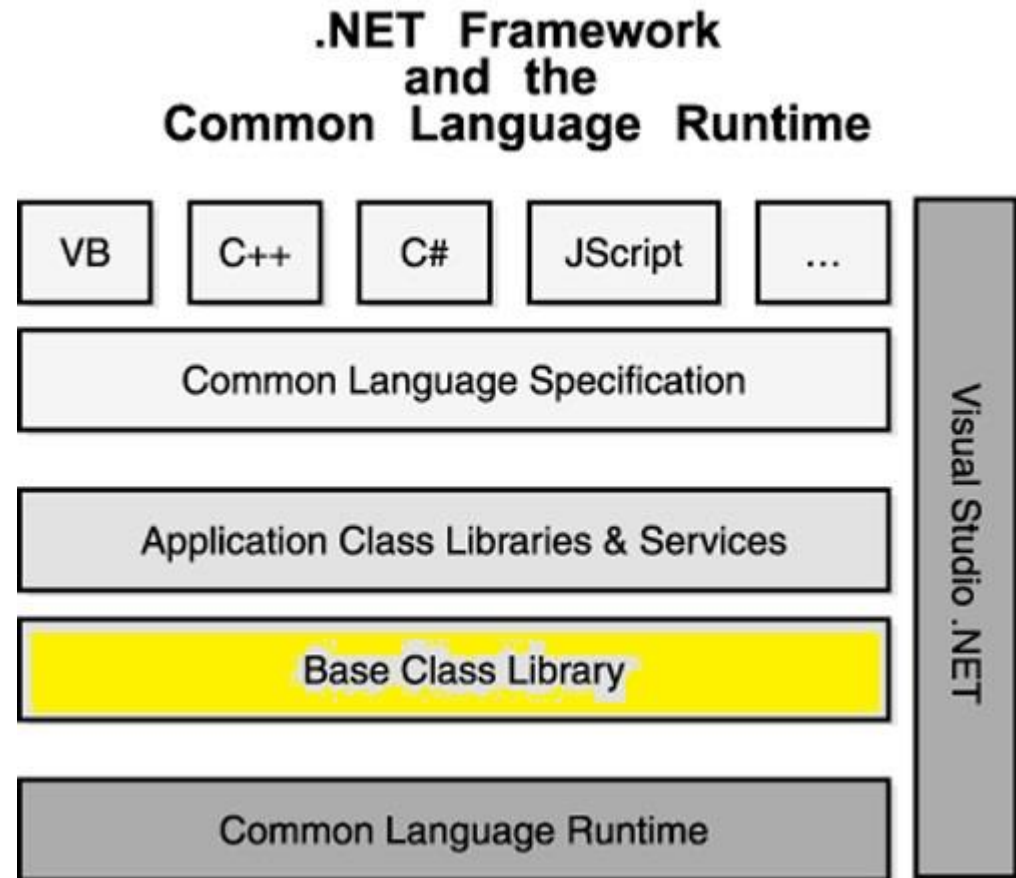
TAREA # 1 Tipos de datos utilizados en C#:

1. Realizar un programa que muestre los diferentes tipos de datos utilizados en C#, utilizar variables y constantes como parte del programa y realizar conversiones entre los mismos.
2. Subir el programa a github o bitbucket y enviar por email (daniel_nuno@hotmail.com) la liga al mismo.

Unidad II, Introducción a Microsoft Visual C#

2.3 La biblioteca de clases

Base Class Library (BCL)



Unidad II, Introducción a Microsoft Visual C#

2.3 La biblioteca de clases

La BCL está constituida por espacios de nombres (**namespaces**). Cada espacio de nombres contiene tipos que se pueden utilizar en el programa: clases, estructuras, enumeraciones, delegados e interfaces.

Cuando se crea un proyecto de Visual C# en Visual Studio, se sigue haciendo referencia a las DLL más comunes de la clase base, pero, si necesita usar un tipo incluido en una DLL a la que aún no se hace referencia, deberá agregar la referencia de esa DLL.

La plataforma .NET incluye una colección de clases bien organizada cuya parte independiente del sistema operativo ha sido propuesta para su estandarización.

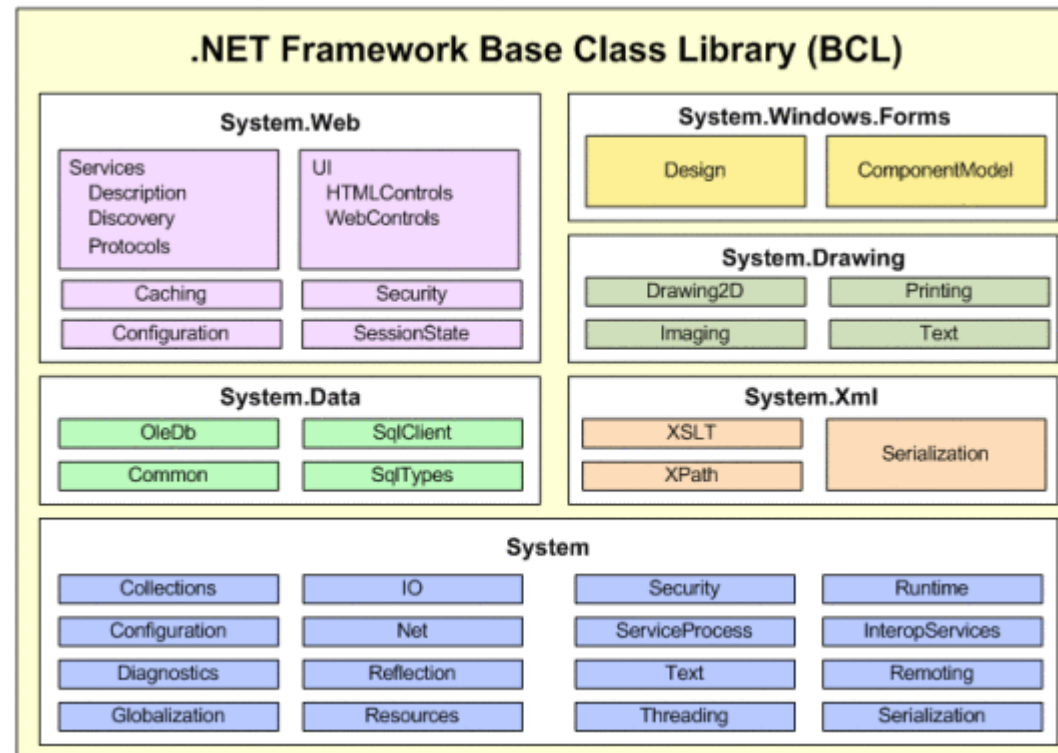
La BCL integra todas las tecnologías Windows en un marco único para todos los lenguajes de programación (Windows Forms, GDI+, Web Forms, Web Services, impresión, redes...).

La BCL proporciona un modelo orientado a objetos que sustituye a los componentes COM.

Unidad II, Introducción a Microsoft Visual C#

- System namespace
 - Namespace raíz para tipos de datos de .NET
 - Contiene clases que representan los tipos de datos base: Object, Byte, Char, Array, Int32, String
 - Contiene mas de 100 clases que van de un rango para manejo de excepciones y hasta ejecución.
 - <https://docs.microsoft.com/en-us/dotnet/api/system>
 - Además, contiene “namespaces” de segundo nivel.
 - <https://docs.microsoft.com/dotnet/api>

Unidad II, Introducción a Microsoft Visual C#



Unidad II, Introducción a Microsoft Visual C#

2.4 Operadores

- Operadores
 - Aritméticos
 - Comparación
 - Asignación
 - Lógicos
 - “Bitwise”
 - Misceláneos

Unidad II, Introducción a Microsoft Visual C#

- Operadores Aritméticos

Significado	Operador	Ejemplo
Add	+	$a + b$
Subtract	-	$a - b$
Multiply	*	$a * b$
Divide	/	a / b
Reminder	%	$a \% b$

Unidad II, Introducción a Microsoft Visual C#

- Operadores Aritméticos (unary)

Significado	Operador	Ejemplo	Equivalencia
Incrementar (post)	++	a++	int a = 1; int b = a++; a = 2, b = 1;
Decrementar (post)	--	a--	int a = 1; int b = a--; a = 0, b = 1;
Incrementar (pre)	++	++a	
Decrementar (pre)	--	--a	

Unidad II, Introducción a Microsoft Visual C#

- Operadores de Comparación

Comparación	Operador	Ejemplo
Equal	==	a==b
Not Equal	!=	a != b
Greater Than	>	a > b
Greater or equal to	>=	a >= b
Less than	<	a < b
Less than or equal to	<=	a <= b

Unidad II, Introducción a Microsoft Visual C#

- Operadores de Asignación

Significado	Operador	Ejemplo	Same as
Assignment	=	a=1	
Addition assignment	+=	a+=3	a=a + 3
Subtraction assignment	-=	a-=3	a=a - 3
Multiplication assignment	*=	a *= 3	a = a * 3
Division assignment	/=	a /= 3	a = a / 3

Unidad II, Introducción a Microsoft Visual C#

- Operadores Lógicos

Significado	Operador	Ejemplo
And	&&	a && b
Or		a b
Not	!	a != b

Unidad II, Introducción a Microsoft Visual C#

- Operadores “Bitwise”

Operador	Descripción	Ejemplo
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, which is 0000 1100
	Binary OR Operator copies a bit if it exists in either operand.	(A B) = 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = 61, which is 1100 0011 in 2's complement due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15, which is 0000 1111

Unidad II, Introducción a Microsoft Visual C#

- Operadores Misceláneos

Operador	Descripción	Ejemplo
sizeof()	Returns the size of a data type.	sizeof(int), returns 4.
typeof()	Returns the type of a class.	typeof(StreamReader);
&	Returns the address of an variable.	&a; returns actual address of the variable.
*	Pointer to a variable.	*a; creates pointer named 'a' to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y
is	Determines whether an object is of a certain type.	If(Ford is Car) // checks if Ford is an object of the Car class.
as	Cast without raising an exception if the cast fails.	Object obj = new StreamReader("Hello"); StreamReader r = obj as StreamReader;

Unidad II, Introducción a Microsoft Visual C#

- Precedencia y asociación de operadores

Category (By Precedence)	Operator(s)	Associativity
Unary	+ - ! ~ ++ -- (type)* & sizeof	Right to Left
Additive	+ -	Left to Right
Multiplicative	% / *	Left to Right
Relational	< > <= >=	Left to Right
Shift	<< >>	Left to Right
Equality	== !=	Right to Left
Logical AND	&	Left to Right
Logical OR		Left to Right
Logical XOR	^	Left to Right
Conditional OR		Left to Right
Conditional AND	&&	Left to Right
Null Coalescing	??	Left to Right
Ternary	?:	Right to Left
Assignment	= *= /= %= += -= <<= >>= &= ^= = =>	Right to Left

Unidad II, Introducción a Microsoft Visual C#

TAREA # 2 Operadores en C#:

1. Realizar un programa que muestre los diferentes tipos de operadores utilizados en C#.
2. Subir el programa a github o bitbucket y enviar por email (daniel_nuno@hotmail.com) la liga al mismo.

Unidad II, Introducción a Microsoft Visual C#

2.5 Instrucciones de control de programa

- Control de Flujo

- Condicionales

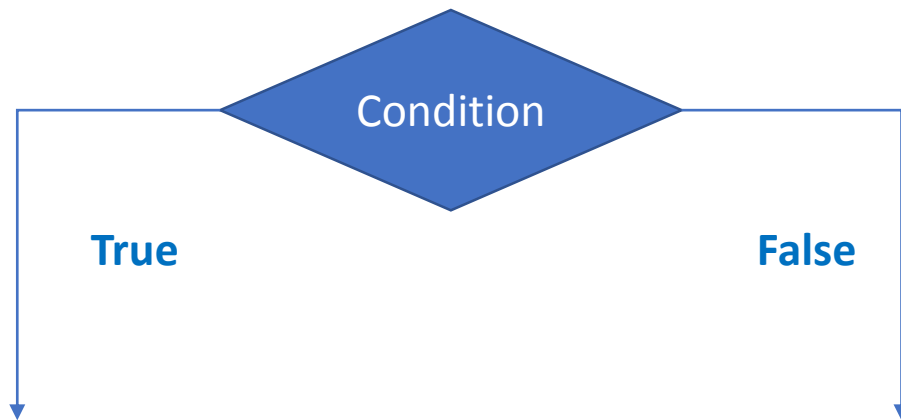
- If/else
 - switch/case
 - Operator a ? b : c

- Iteración

- Ciclo “for”
 - Ciclo “foreach”
 - Ciclo “while”
 - Ciclo “do-while”

Unidad II, Introducción a Microsoft Visual C#

- Control de Flujo
 - Condicionales
 - if/else



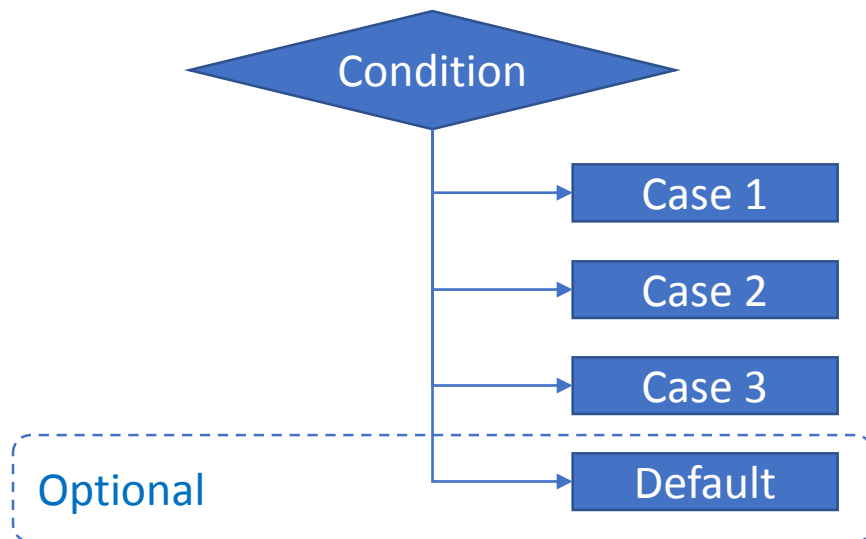
```
if(boolean_expression) {  
    /* statement(s) will execute if the boolean expression  
    is true */  
}
```

```
if(boolean_expression) {  
    /* statement(s) will execute if the boolean expression  
    is true */  
} else {  
    /* statement(s) will execute if the boolean expression  
    is false */  
}
```

```
if( boolean_expression 1) {  
    /* Executes when the boolean expression 1 is true */  
    if(boolean_expression 2) {  
        /* Executes when the boolean expression 2 is true */  
    }  
}
```

Unidad II, Introducción a Microsoft Visual C#

- Control de Flujo
 - Condicionales
 - switch/case



```
switch(expression) {  
    case constant-expression :  
        statement(s);  
        break; /* optional */  
    case constant-expression :  
        statement(s);  
        break; /* optional */
```

```
    /* you can have any number of case  
statements */  
    default : /* optional */  
        statement(s);  
}
```

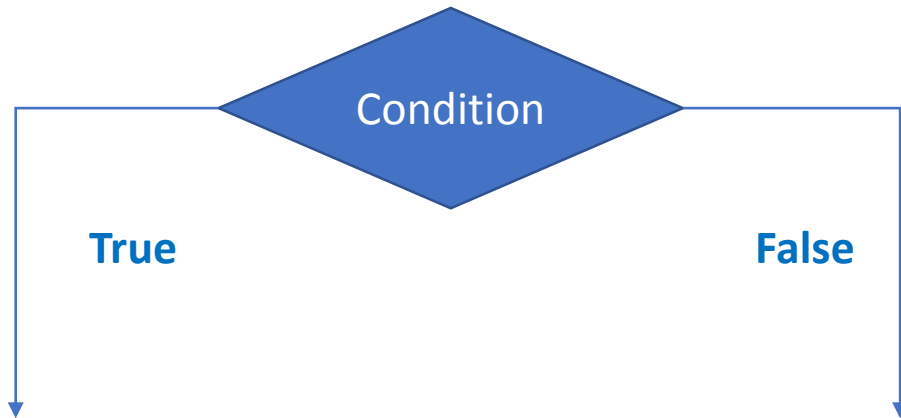
Unidad II, Introducción a Microsoft Visual C#

- Control de Flujo

- Condicionales

- `Exp1 ? Exp2 : Exp3;`

`Exp1 ? Exp2 : Exp3;`



Unidad II, Introducción a Microsoft Visual C#

- Control de Flujo

- Iteración

- Ciclo “for”
 - Ciclo “foreach”

```
for ( init; condition; increment ) {  
    statement(s);  
}
```

```
foreach (var number in numbers) {  
    statement(s);  
}
```

Unidad II, Introducción a Microsoft Visual C#

- Control de Flujo
 - Iteración
 - Ciclo “while”
 - Ciclo “do-while”
 - Se ejecuta al menos una vez

```
while(condition) {  
    statement(s);  
}
```

```
do {  
    statement(s);  
} while( condition );
```


Unidad II, Introducción a Microsoft Visual C#

2.6 Clases, objetos y métodos en C#.

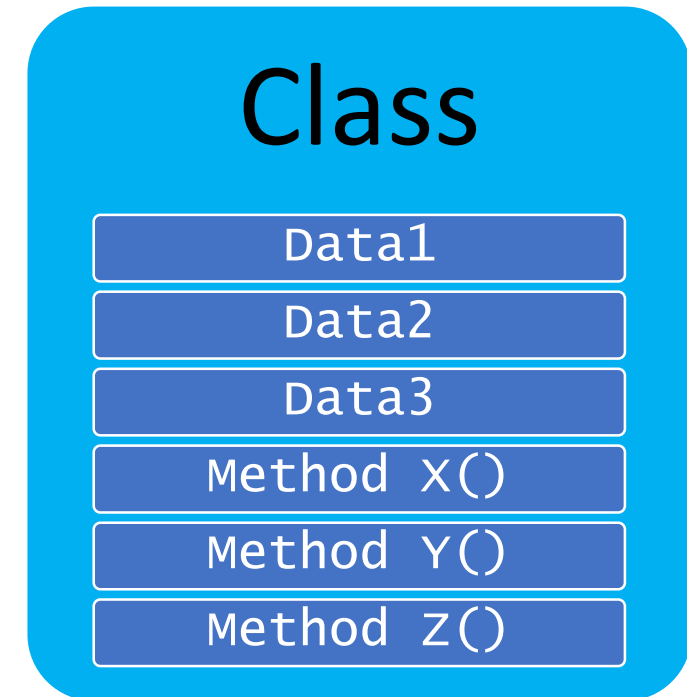
- Non-Primitive Types
 - Classes
 - Structs
 - Arrays
 - Strings
 - Enums

Unidad II, Introducción A Microsoft Visual C#

Non-Primitive Types

Clases

- Una clase es una **plantilla (molde)**, que define ***atributos*** (variables) y ***métodos***(funciones)
- La clase **define los atributos y métodos** comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.
- ***Debemos crear una clase*** antes de poder crear objetos (***instancias***) de esa clase. Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto propiamente dicho.



Unidad II, Introducción A Microsoft Visual C#

Clases

- La sintaxis de una clase en C# es:

```
class[nombre de la clase] {  
  [atributos o variables de la clase]  
  [métodos o funciones de la clase]  
  [main]  
}
```

- Un **método** es un *conjunto de instrucciones* a las que se les asocia un nombre de modo que si se desea ejecutarlas basta referenciarlas a través de dicho nombre en vez de tener que escribirlas.
- Dentro de estas instrucciones es posible acceder con total libertad a la información almacenada en los campos pertenecientes a la clase dentro de la que el método se ha definido, *los métodos permiten manipular los datos almacenados en los objetos.*

Unidad II, Introducción A Microsoft Visual C#

Clases y constructores

- *Cada vez que se crea una clase se llama a su **constructor***
- Un constructor es un método que se utiliza para inicializar la clase.
- Una clase puede tener **varios constructores** que toman argumentos diferentes.
- Los **constructores** permiten al programador establecer valores predeterminados, limitar la creación de instancias y escribir código flexible y fácil de leer.

Unidad II, Introducción A Microsoft Visual C#

Características de los Constructores

- Tiene el mismo nombre de la clase.
- Es el primer método que se ejecuta.
- Se ejecuta en forma automática.
- No puede retornar datos.
- Se ejecuta una única vez.
- Un constructor tiene por objetivo inicializar atributos.

• SINTAXIS

```
Modificador NombredeLaClase(Parámetros)
{
    Instrucciones
}
```

Unidad II, Introducción A Microsoft Visual C#

Constructores

- Las clases pueden definir constructores que acepten parámetros.
- Las clases pueden definir varios constructores y no se requiere ninguno para definir un constructor predeterminado.
- Se debe llamar a constructores que toman parámetros a través de una instrucción “new”.
- Al igual que los métodos, los constructores también pueden ser sobrecargados.

```
public class Employee
{
    public int salary;

    public Employee(int annualSalary)
    {
        salary = annualSalary;
    }

    public Employee(int weeklySalary, int numberOfWeeks)
    {
        salary = weeklySalary * numberOfWeeks;
    }
}
```

```
Employee e1 = new Employee(30000);
Employee e2 = new Employee(500, 52);
```

Unidad II, Introducción A Microsoft Visual C#

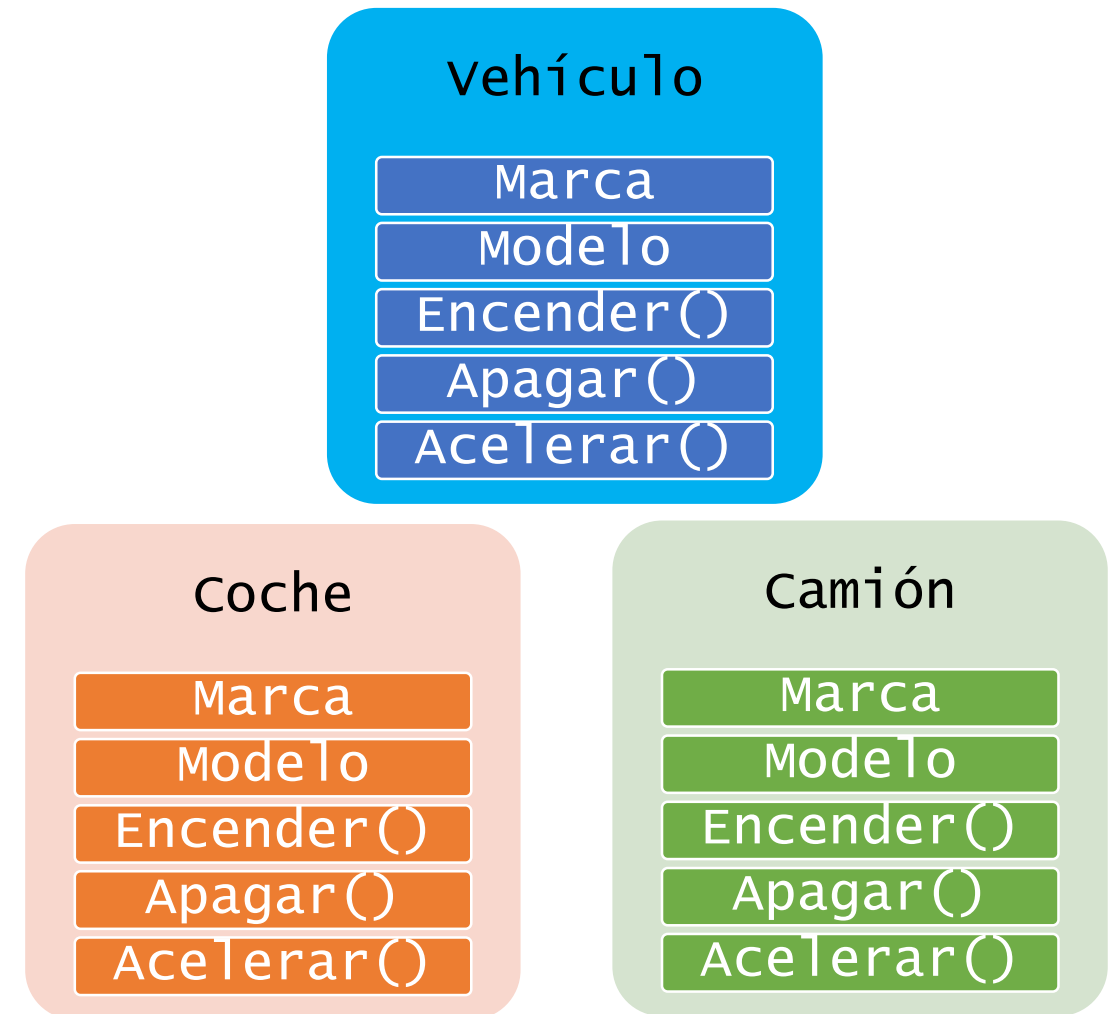
Destruyores ([finalizers](#))

- Un destructor se declara como un constructor, aunque va precedido por un signo de tilde ~.
- Se emplea una des asignación de memoria de objetos no referenciados (recolección de basura), y cuando esto ocurre se ejecuta el destructor de dicha clase.
- El destructor de una clase no se llama cuando un objeto sale del ámbito.
- Todos los destructores se llamarán antes de que finalice un programa.
- La palabra clave “**this**” es un apuntador al mismo objeto en el cual se usa.
- C# permite la sobrecarga de operadores con la palabra clave **operator**

Unidad II, Introducción A Microsoft Visual C#

Herencia

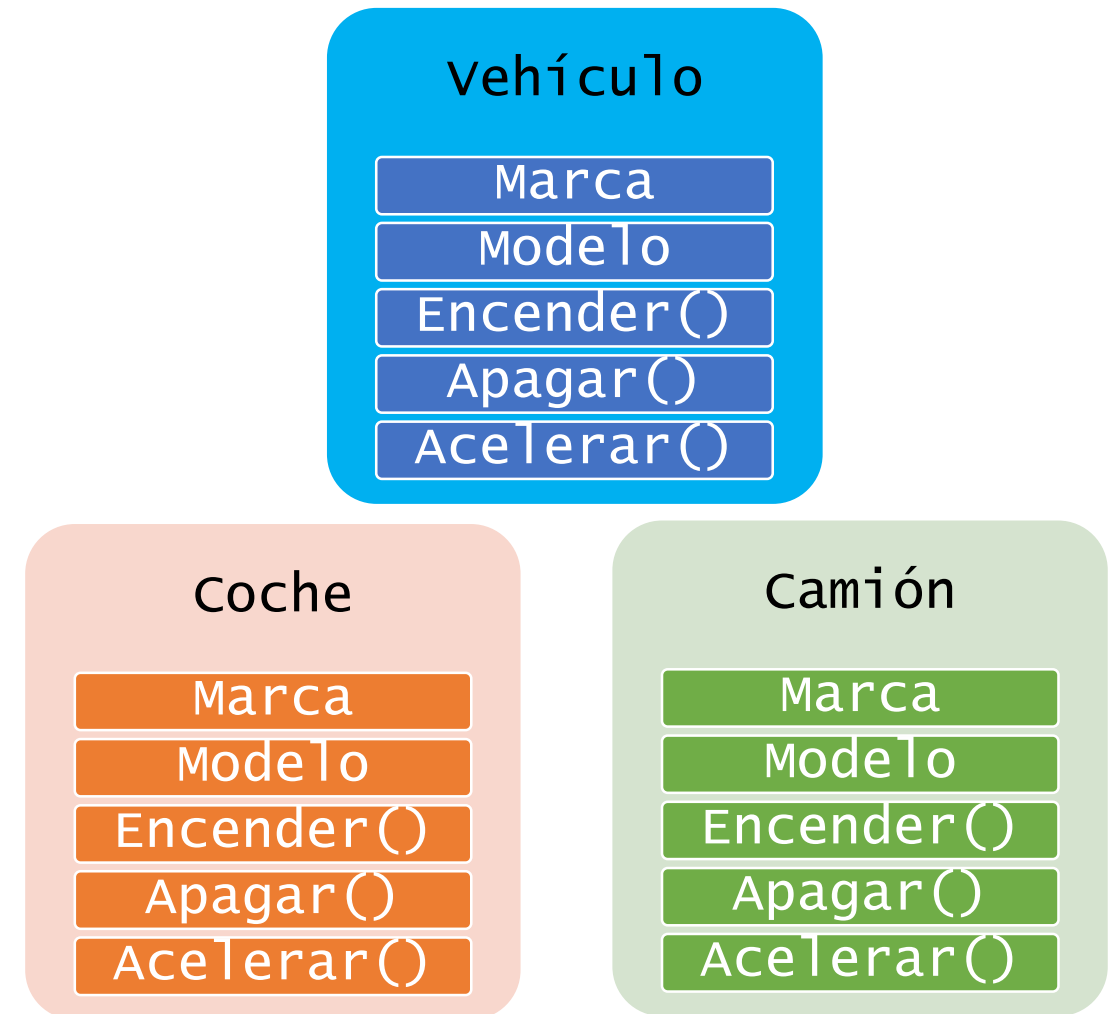
- **La herencia** nos permite derivar una nueva clase a partir de una existente.
- La clase existente es conocida como clase madre, clase padre, o superclase, o clase base.
- La clase derivada también es conocida como clase hija, o subclase.
- Una clase puede ser derivada de más de una clase o interface, lo que quiere decir que puede heredar datos y métodos de múltiples clases base.



Unidad II, Introducción A Microsoft Visual C#

Herencia

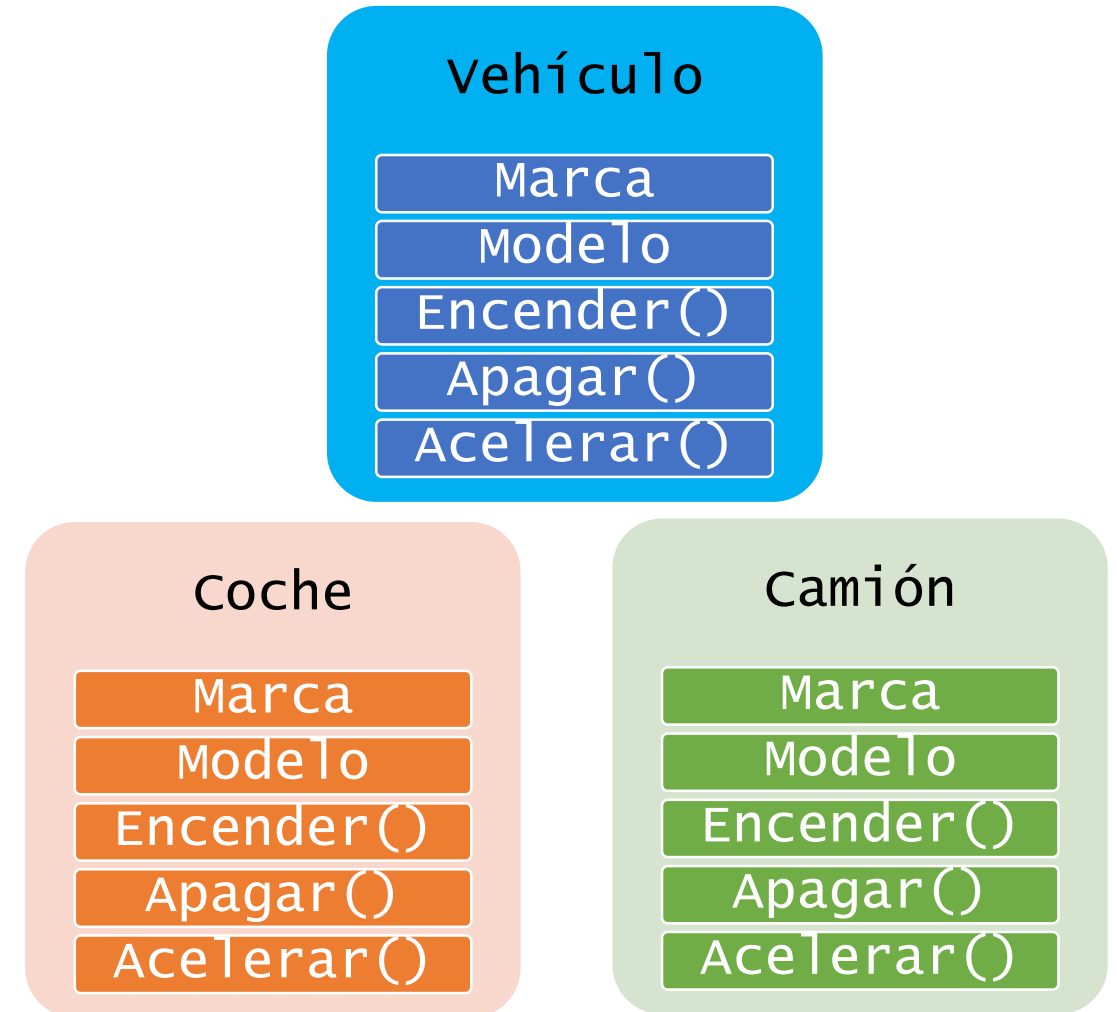
- **La herencia es la columna vertebral de la POO.** Permite a los programadores crear una jerarquía entre un grupo de clases que tienen características similares.
- La herencia **es una forma de reutilización de código.**



Unidad II, Introducción A Microsoft Visual C#

Encapsulamiento

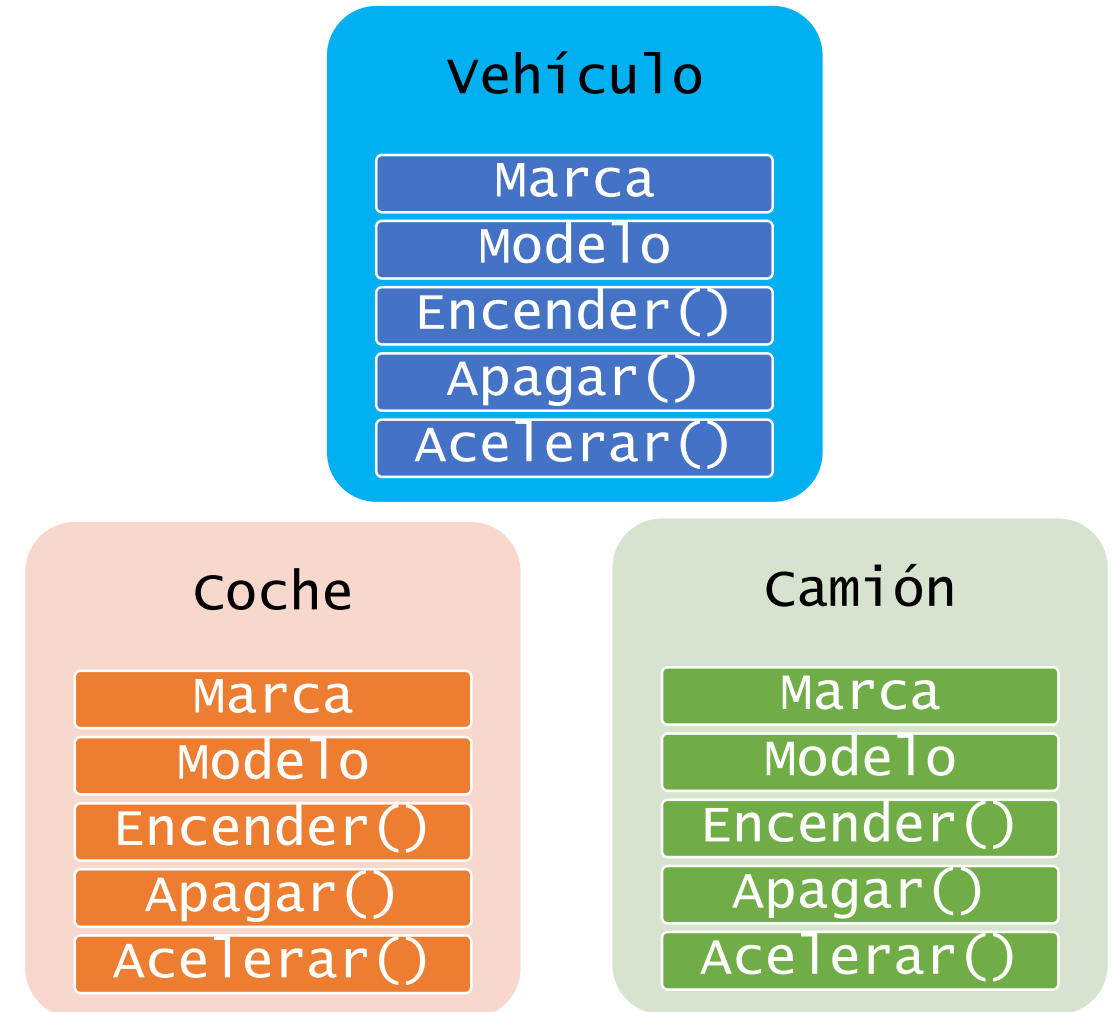
- Es la propiedad que tienen los objetos, de contener tanto datos como métodos, los cuales pueden manipular o cambiar estos datos.
- **Consiste en separar los aspectos externos de un objeto**(que pueden ser accedidos desde otros objetos) **de los detalles internos** de implementación del mismo.



Unidad II, Introducción A Microsoft Visual C#

Encapsulamiento

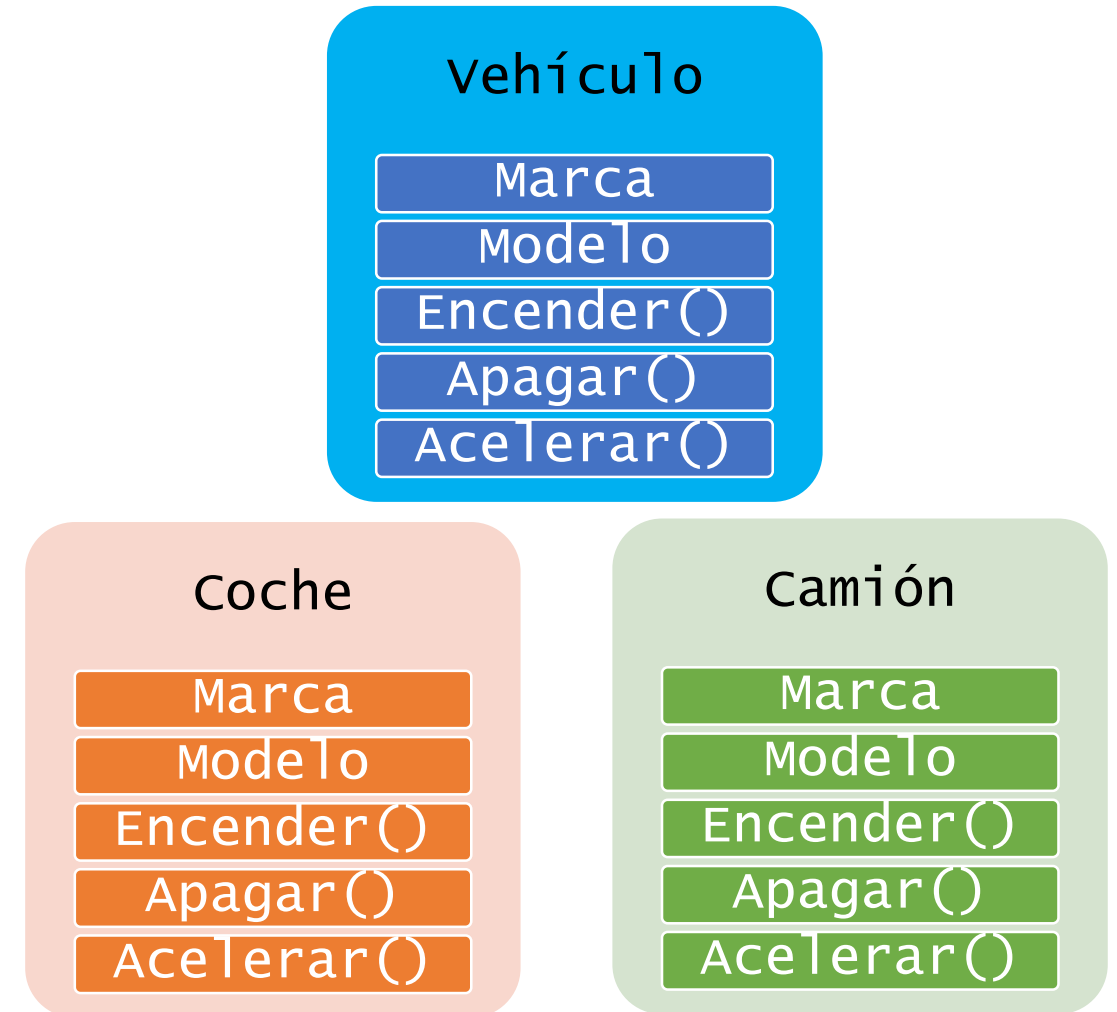
- Mediante esta propiedad, los objetos, tienen el control necesario, de la integridad de los datos contenidos en estos.
- Los clientes de una clase sólo conocen la interfaz de la misma, es decir, conocen los prototipos de las operaciones pero no cómo están implementadas



Unidad II, Introducción A Microsoft Visual C#

Encapsulamiento

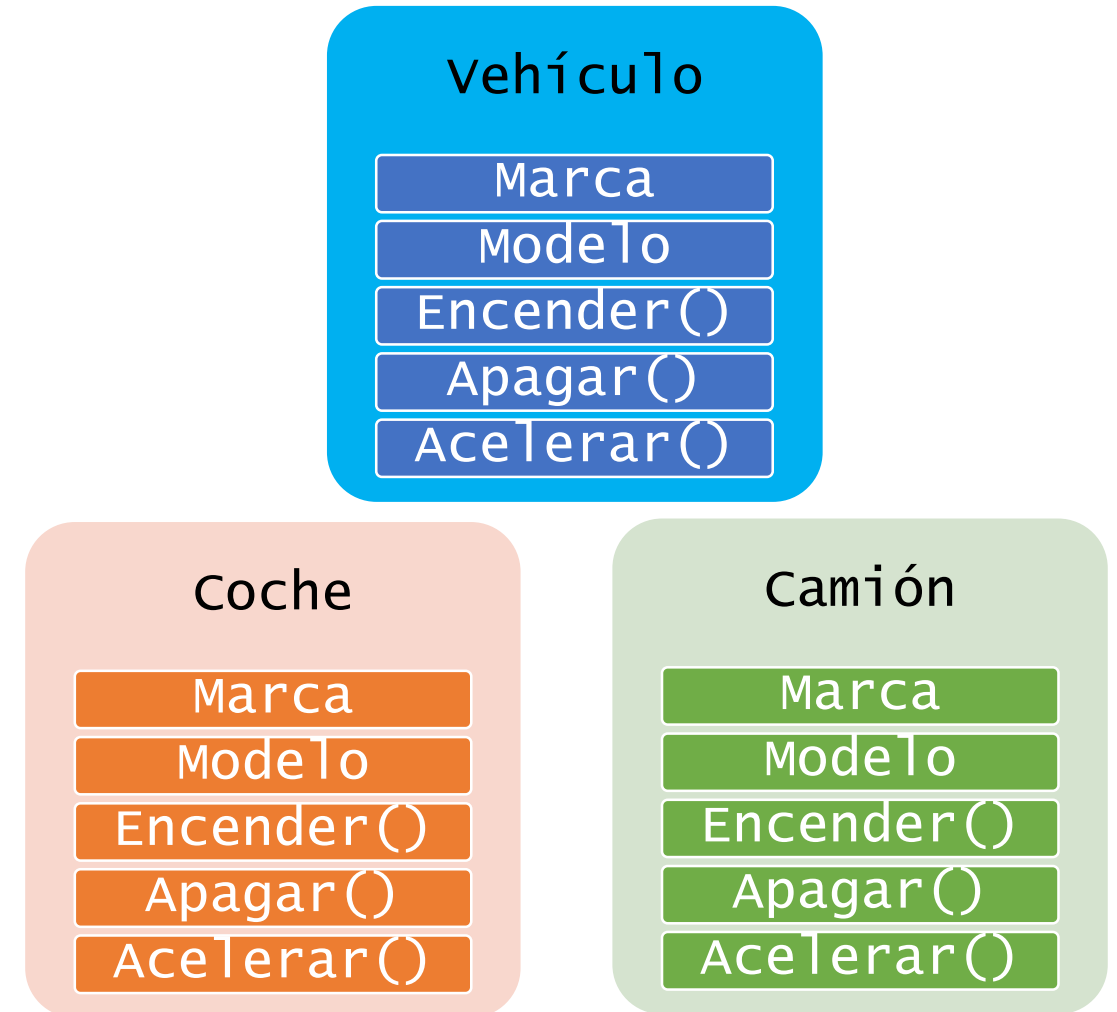
- Se realiza el encapsulamiento mediante el uso de modificadores de acceso:
 - Public
 - Private
 - Protected
 - Internal
 - Protected Internal



Unidad II, Introducción A Microsoft Visual C#

Polimorfismo

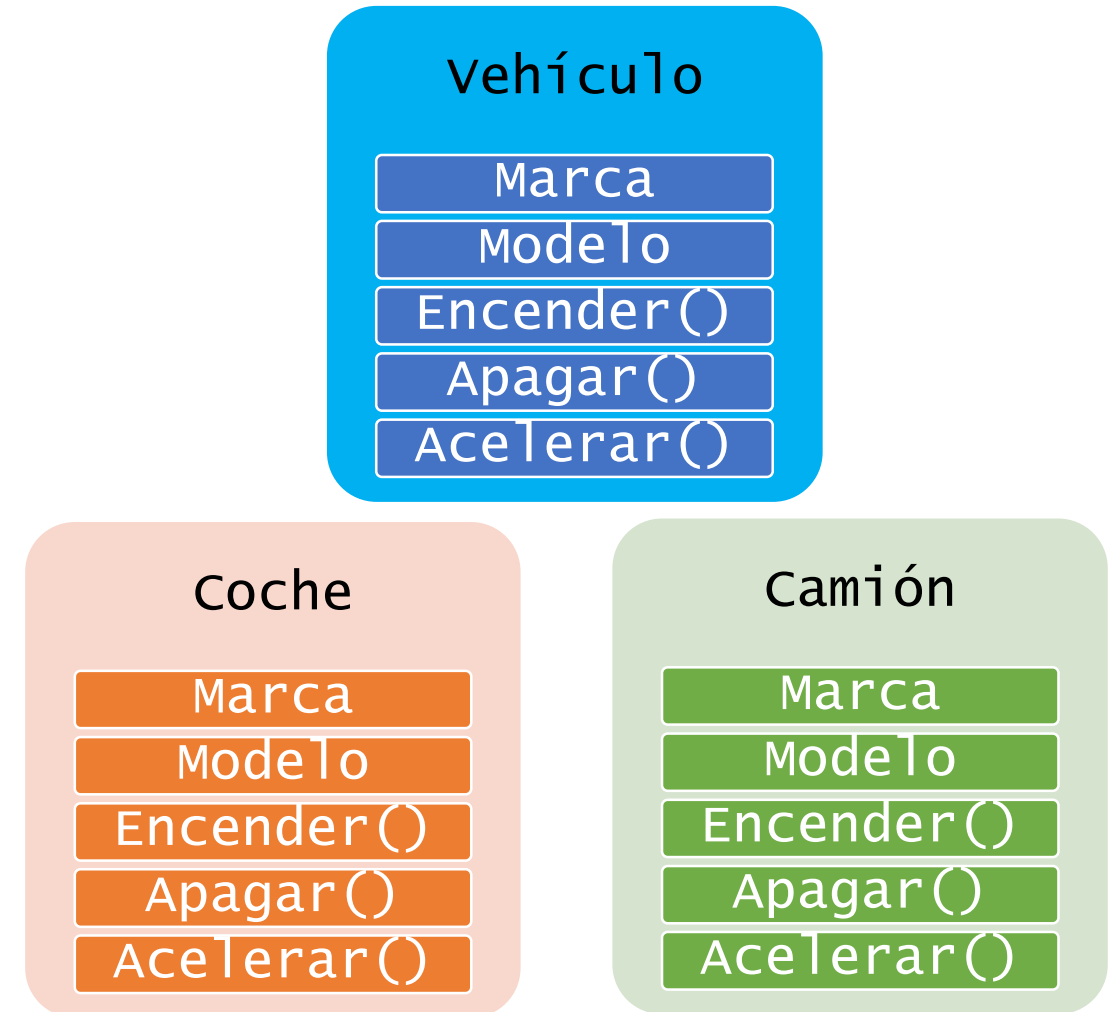
- Significa que la misma operación puede comportarse diferente en clases distintas.
- El polimorfismo está muy ligado a la herencia.
- Distintas instancias del mismo tipo interpretan el mismo mensaje en diferentes formas.



Unidad II, Introducción A Microsoft Visual C#

Polimorfismo

- El polimorfismo requiere enlace dinámico
- **Polimorfismo dinámico:** la llamada se resuelve en tiempo de ejecución.
- **Polimorfismo estático:** la llamada se resuelve en tiempo de compilación.



Unidad II, Introducción a Microsoft Visual C#

TAREA # 3 Clases en C#:

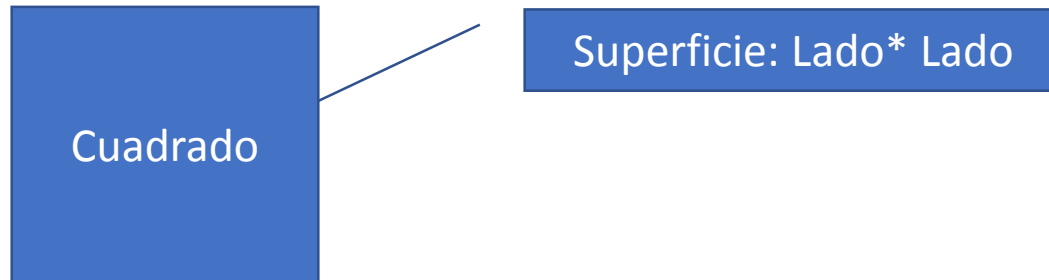
1. Realizar un programa que muestre el uso de Clases en C#, se puede reusar el código ya hecho en la tarea 1 y 2.
2. Subir el programa a github o bitbucket y enviar por email (daniel_nuno@hotmail.com) la liga al mismo.

Unidad II, Introducción a Microsoft Visual C#

Ejercicio # 3 en clase:

Clases en C#:

1. Desarrollar un programa que tenga una clase que represente un Cuadrado y tenga los siguientes métodos:
2. ingresar valor a su lado,
3. imprimir su perímetro y su superficie.
4. Subir el programa a github o bitbucket y enviar por email (daniel_nuno@hotmail.com) la liga al mismo.



Unidad II, Introducción a Microsoft Visual C#

Non-Primitive Types

Structs

- Sirven para que una variable, pueda tener información relacionada.
- Se utilizan para representar un registro.
- No pueden recibir herencia de otras estructuras.

```
public struct RGBColor
{
    public int Red;
    public int Green;
    public int Blue;
}
```

Unidad II, Introducción a Microsoft Visual C#

Structs

- No pueden ser usadas como base de otras estructuras
- Las estructuras pueden tener un constructor pero no un destructor
- El constructor es definido automáticamente y no se puede modificar
- Ideales para definir un objeto pequeño, del que se crearían cientos

```
public struct RGBColor
{
    public int Red;
    public int Green;
    public int Blue;
}
```

Unidad II, Introducción a Microsoft Visual C#

Clases vs Structs

- Las clases son tipos de referencia, las estructuras son tipos de valor.
- Las estructuras no soportan herencia.
- Las estructuras no pueden tener un constructor default.
- El constructor en las estructuras es automáticamente definido y no se puede cambiar.

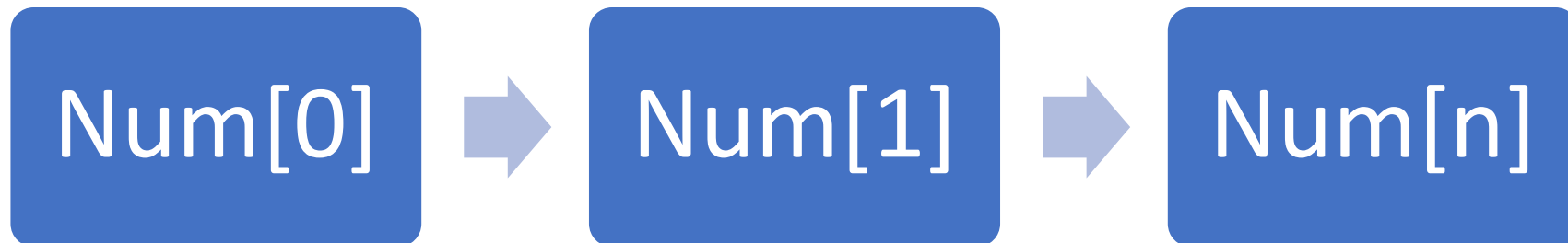
Unidad II, Introducción a Microsoft Visual C#

2.7 Arreglos y Matrices

Non-Primitive Types

Arrays,

- Un arreglo almacena una colección secuencial de elementos de un tamaño fijo y de un mismo tipo.
- Todos los arreglos consisten en localidades de memoria contigua.



Unidad II, Introducción a Microsoft Visual C#

Non-Primitive Types

Arrays, Matrices

- Los índices empiezan en cero.
- Cuando se declara un arreglo, los corchetes [] deben ir después del tipo, no después del identificador.
- Los arreglos no se inicializan automáticamente.

- Declaración válida:

```
datatype[] arrayName;
```

Ejemplo:

```
int[] valores;
```

- Declaración inválida:

```
int valores[];
```

Unidad II, Introducción a Microsoft Visual C#

Non-Primitive Types

Arrays, Matrices

- Los arreglos es una clase de referencia **System.Array**, se requiere utilizar la palabra “new” para instanciarlos.
- Se utiliza el índice para acceder a los valores almacenados.
- [Propiedades y métodos de System.Array](#)

- Declaración válida:

```
datatype[] arrayName;
```

Ejemplo:

```
int[] valores;
```

- Declaración inválida:

```
int valores[];
```

Unidad II, Introducción a Microsoft Visual C#

Non-Primitive Types

Arreglos Multidimensionales,

- Los arreglos multidimensionales son realmente objetos **System.Array** que es el tipo base abstracto de todos los tipos de arreglos.
- Las propiedades y otros miembros de la clase **System.Array** se pueden utilizar cuando sea necesario.
- Ejem:
 - La propiedad *Length* para obtener la longitud de una matriz.

Unidad II, Introducción a Microsoft Visual C#

Non-Primitive Types

Arreglos Multidimensionales,

- Para declarar un arreglo multidimensional
- Dos Dimensiones: `datatype[,] arrayName;`
- Tres Dimensiones: `datatype[, ,] arrayName;`

• Jagged Arrays

- Arreglos de Arreglos
- `datatype[][] arrayName;`

Unidad II, Introducción a Microsoft Visual C#

Non-Primitive Types

Arreglos como parámetros

- Los arreglos pueden ser pasados como parámetros entre funciones.
- Facilitan el pasar argumentos cuando se desconoce la cantidad de los mismos.

```
class ParamArray {  
    public int AddElements(params int[] arr) {  
        // Logic  
    }  
  
int suma = AddElements(512, 720, 250, 567, 889);
```

Unidad II, Introducción a Microsoft Visual C#

- Non-Primitive Types
 - ~~Classes~~
 - ~~Structs~~
 - ~~Arrays~~
 - Strings
 - Enums

Unidad II, Introducción a Microsoft Visual C#

- Non-Primitive Types

- Strings

- Los strings se pueden utilizar como un arreglo de caracteres
 - Se puede utilizar también, la palabra reservada “string” para declarar una variable
 - La palabra reservada “string” es un alias de la clase **System.String**
 - Los strings son inmutables, es decir, que no se pueden cambiar, existen funciones que pueden manipularlos, pero el string resultante, será un nuevo string.

- Propiedades y métodos de System.String

Unidad II, Introducción a Microsoft Visual C#

- Non-Primitive Types
 - Strings
- **Se puede crear un “string” de la siguientes maneras:**
 - Al asignar un literal de cadena a una variable de cadena
 - Mediante el uso de un constructor de clase String
 - Mediante el uso del operador de concatenación de cadenas (+)
 - Recuperando una propiedad o llamando a un método que devuelve una cadena
 - Llamando a un método de formateo para convertir un valor o un objeto a su representación de cadena

Unidad II, Introducción a Microsoft Visual C#

- Non-Primitive Types
 - Strings
 - Caracteres de escape

Escape Character	Description
\n	New line
\t	Tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark

Unidad II, Introducción a Microsoft Visual C#

- Non-Primitive Types

- Strings

- Verbatim strings

- `string path = "c:\\proj\\proj1\\folder1";`

- `string path = @"c:\proj\proj1\folder1";`

Unidad II, Introducción a Microsoft Visual C#

- Non-Primitive Types

- Enums

- Un grupo de pares que consisten en un nombre y un valor.
 - Son constantes, no pueden ser cambiadas.
 - No pueden ser heredados
 - No pueden heredar

```
enum <enum_name> {  
    enumeration list  
};
```

Ejemplo:

```
enum Days { Sun, Mon, tue,  
Wed, thu, Fri, Sat };
```

Unidad II, Introducción a Microsoft Visual C#

```
const int RegularMail = 1;  
const int RegisterMail = 2;  
const int ExpressMail = 3;
```

```
public enum MailType : byte  
{  
    RegularMail = 1,  
    RegisterMail = 2,  
    ExpressMail = 3  
};
```

```
var method =  
MailType.ExpressMail;
```


Unidad II, Introducción a Microsoft Visual C#

- **Value Types**

- Structures

 - Primitive Types

 - int
 - char
 - float
 - bool

 - Custom Structures

- Allocated on stack
- Memory allocation automatically
- Immediately removed when out of scope

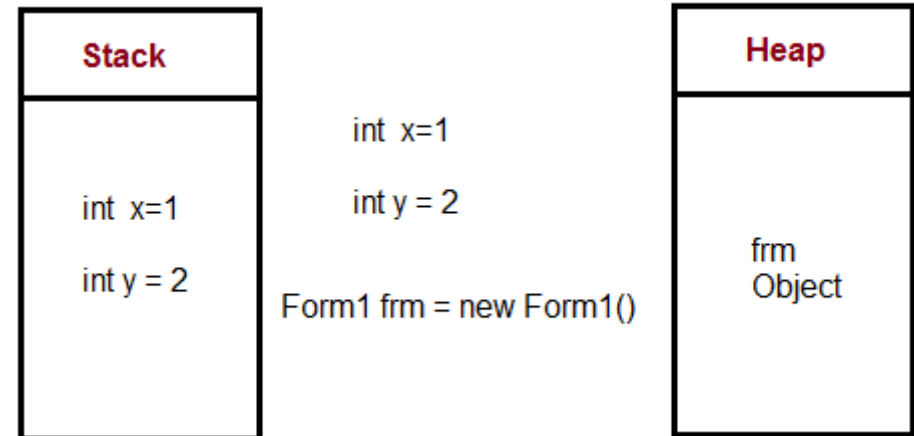
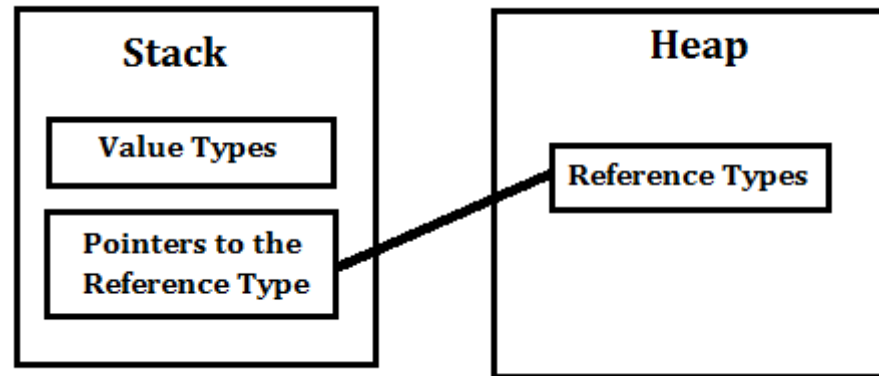
- **Reference Types**

- Classes

 - Array (System.Array)
 - Strings (System.String)
 - Custom classes

- You need to allocate memory
- Memory allocated on the heap
- Garbage collector

Unidad II, Introducción a Microsoft Visual C#



Unidad II, Introducción a Microsoft Visual C#

Ejercicio # 4 en clase:

- Realizar un programa que haga un recorrido en una matriz dada e imprima el contenido:

1	9	23	34	102
2	8	67	56	23
3	7	84	78	43
4	5	90	98	34

Resultado: 1,2,3,4,5,7,8,9,23,67,84,90,98,78,56,34,102,23,43,34

- Subir el programa a github o bitbucket y enviar por email (daniel_nuno@hotmail.com) la liga al mismo.

Unidad II, Introducción a Microsoft Visual C#

Ejercicio # 5 en clase:

1) Entrar a la página de Microsoft DeveloperNetwork MSDN:

<https://msdn.microsoft.com/es-mx>>Documentación > API y referencia >

**2) En la página ubicar la sección de: Herramientas y Lenguajes de Desarrollo
>Seleccionar: Visual Studio 2015**

3) Con apoyo de la herramienta de búsqueda ubicar las siguientes páginas de contenido:

Tutorial: [Crear una aplicación sencilla con Visual C# o Visual Basic](#)

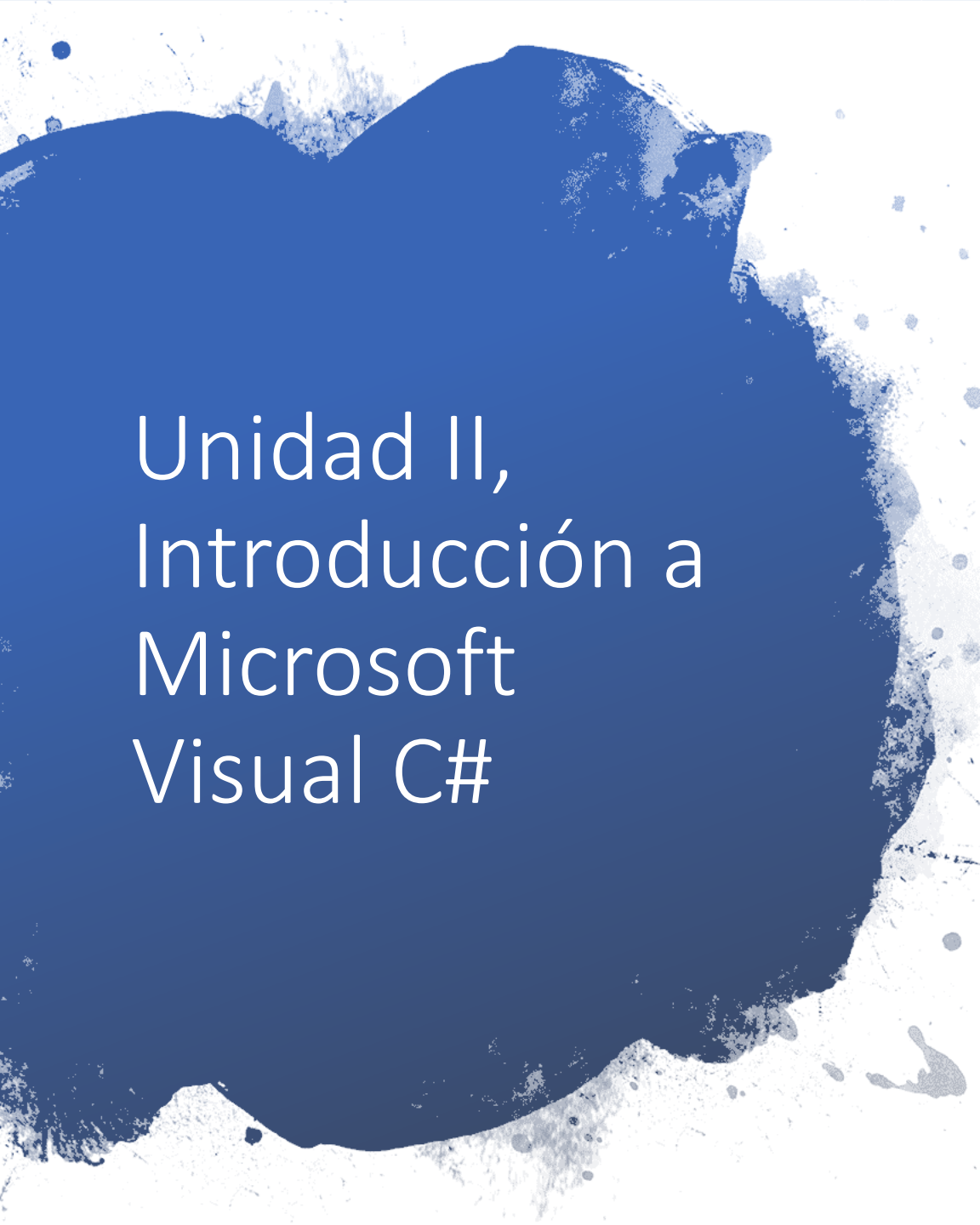
4) Seguir las instrucciones para crear la aplicación

**5) Subir el programa a github o bitbucket y enviar por email
(daniel_nuno@hotmail.com) la liga al mismo.**

Unidad II, Introducción a Microsoft Visual C#

TAREA # 4 palíndromo en C#:

1. Realizar un programa que haga diga si una cadena es un “palíndromo”.
 1. ¿Acaso hubo búhos acá?
 2. A la catalana banal, atácala.
 3. A mamá, Roma le aviva el amor a papá y a papá, Roma le aviva el amor a Mamá.
 4. A ti no, bonita.
 5. Amo la pacífica paloma.
 6. Amor a Roma.
 7. Ana lava lana.
2. Subir el programa a github o bitbucket y enviar por email (daniel_nuno@hotmail.com) la liga al mismo.



Unidad II, Introducción a Microsoft Visual C#

~~2.1 Breve descripción del C#.~~

~~2.2 Tipos de datos básicos.~~

~~2.3 La biblioteca de clases.~~

~~2.4 Operadores.~~

~~2.5 Instrucciones de control de programa.~~

~~2.6 Clases, objetos y métodos en C#.~~

~~2.7 Arreglos y Matrices, Cadenas.~~

2.8 Sobrecarga de operadores.

2.9 Indizadores y propiedades.

2.10 Herencia en C#.

2.11 Interfaces, ~~estructuras y~~
~~enumeraciones.~~

2.12 Colecciones (listas, hash, stack, queue)

Unidad II, Introducción a Microsoft Visual C#

2.8 Sobrecarga de operadores

- Se puede redefinir o sobrecargar la mayoría de los operadores.
- Los operadores sobrecargados son funciones con un nombre especial que incluye la palabra “operator” seguida del operador que esta siendo definido.
- Un operador sobrecargado regresa un tipo de dato y acepta parámetros.

Ver ejemplo:

https://www.tutorialspoint.com/csharp/csharp_operator_overloading.htm

Unidad II, Introducción a Microsoft Visual C#

2.8 Sobrecarga de operadores

- C# permite que los tipos definidos por el usuario sobrecarguen operadores al definir funciones miembro estáticas mediante la palabra clave [operator](#).
- No todos los operadores se pueden sobrecargar y algunos presentan restricciones.
- <https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/statements-expressions-operators/overloadable-operators>

Unidad II, Introducción a Microsoft Visual C#

2.9 Índices y propiedades.

Un índice permite que un objeto sea indexado como un arreglo.

Cuando se define un índice para una clase, esta clase se comporta similar a un arreglo virtual.

Entonces se puede acceder a la instancia de esta clase usando el operador “[]”.

Unidad II, Introducción a Microsoft Visual C#

2.9 Índices y propiedades.

```
element-type this[int index] {  
  
    // The get accessor.  
    get {  
        // return the value specified by index  
    }  
  
    // The set accessor.  
    set {  
        // set the value specified by index  
    }  
}
```

Unidad II, Introducción a Microsoft Visual C#

2.9 Índices y propiedades.

Uso de los índices.

Se pueden utilizar como a las propiedades de una clase, con un “get” y un “set”.

Sin embargo, las propiedades regresan o definen un miembro específico, y los índices regresan o definen un valor particular para la instancia del objeto.

Ejemplo:

https://www.tutorialspoint.com/csharp/csharp_indexers.htm

Unidad II, Introducción a Microsoft Visual C#

2.10 Herencia en C#

La herencia es uno de los conceptos mas importantes de programación orientada a objetos.

Herencia te permite definir una clase en términos de otra clase, lo cual permite crear y mantener una aplicación de manera mas sencilla.

Herencia promueve el reúso de código y mejora el tiempo de implementación.

Unidad II, Introducción a Microsoft Visual C#

2.10 Herencia en C#

Una clase puede ser heredada de únicamente una clase, pero puede heredar de varias interfaces.

Una clase derivada hereda de la clase base los atributos y los métodos, por lo que estos deben ser creados previamente.

Ejemplo:

https://www.tutorialspoint.com/csharp/csharp_inheritance.htm

Unidad II, Introducción a Microsoft Visual C#

2.11 Interfaces, estructuras y enumeraciones.

- Una interface es un plano o un contrato que una clase al usarlo, debe seguir.
- La interface define el que y en la clase que la usa, se define el como.
- Las interfaces definen atributos, métodos y eventos.
- En las interfaces únicamente se realiza la declaración de los miembros, es responsabilidad de la clase que implementa, la definición de los mismos.
- Ejemplo:
- https://www.tutorialspoint.com/csharp/csharp_interfaces.htm

Unidad II, Introducción a Microsoft Visual C#

TAREA # 5 altas, bajas, cambios en C#:

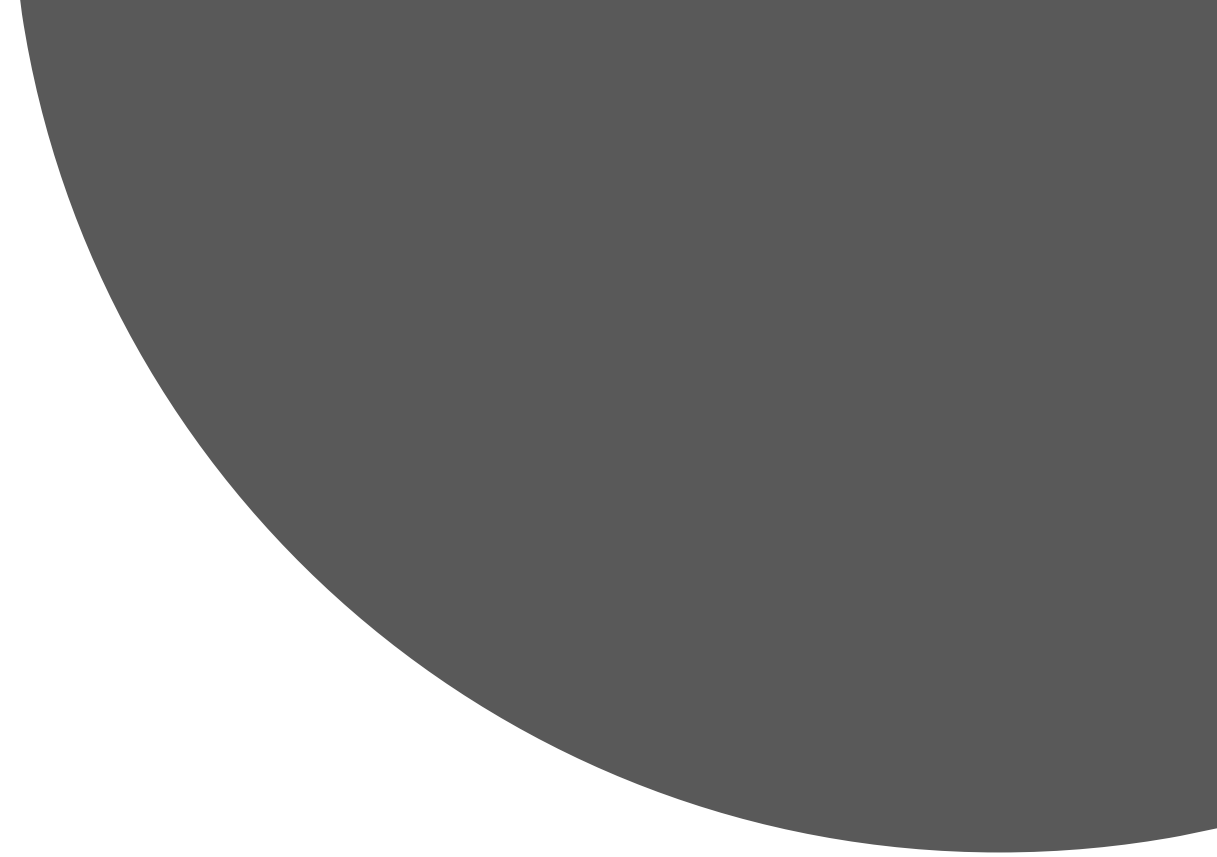
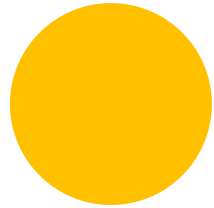
1. Utilizando Indexers e interfaces, hace un programa que haga altas, bajas, cambios de un objeto que puede ser una persona por ejemplo, y tener como atributo su nombre y opcionalmente apellido.

El desarrollo puede ser utilizando WinForms o CommandLine.

La interface debe tener:

```
public interface IABC
void Alta();
void Baja();
void Cambio();
String showElemento();
```

2. Subir el programa a github o bitbucket y enviar por email (daniel_nuno@hotmail.com) la liga al mismo.



Unidad III, Microsoft Visual C# Avanzado



Unidad III, Microsoft Visual C# Avanzado

3.1 Colecciones

3.2 Control de excepciones.

3.3 Entrada y salida.

3.4 DateTime y TimeSpan.

3.5 Espacios de nombres.

3.6 Preprocesador y ensamblados.

3.7 Código no seguro y apuntadores

3.8 Expresiones Lambda

Unidad III, Microsoft Visual C# Avanzado

3.1 Colecciones (list, hash, sorted list, stack, queue, bit array)

- Las colecciones son clases especializadas que se utilizan para manejar varios valores u objetos de diferentes series.
- Existen dos tipos de colecciones:
 - non-generic collections
 - generic collections
- Cada clase de colecciones implementa la interface “[IEnumerable](#)” por lo que los valores se pueden acceder mediante un “**foreach**” .
- El namespace **Sytem.Collections** incluye las siguientes colecciones no genéricas (non-generic): ArrayList, SortedList, Stack, Queue, HashTable, BitArray.

Unidad III, Microsoft Visual C# Avanzado

3.1 Colecciones (list, hash, sorted list, stack, queue, bit array)

Non-generic collection	Uso
ArrayList	Almacena objetos de cualquier tipo a manera de arreglo. No se necesita especificar el tamaño del arreglo, ya que se incrementa automáticamente.
SortedList	Almacena pares donde uno es la llave y el otro es el valor. Automáticamente acomoda los elementos en modo ascendente utilizando la llave.
Stack	Almacena en forma de pila (LIFO). Provee métodos para almacenar, recuperar elementos.
Queue	Almacena en forma de cola (FIFO). Mantiene el orden en el que los elementos fueron agregados. Provee métodos para almacenar y recuperar.
Hashtable	Almacena pares donde uno es la llave y el otro es el valor. Recupera los valores utilizando la llave dada.
BitArray	Arreglo compacto de valores binarios, donde “true” indica un bit en “1”.

Unidad III, Microsoft Visual C# Avanzado

3.1 Colecciones (list, hash, sorted list, stack, queue, bit array)

Ejemplos:

- https://www.tutorialspoint.com/csharp/csharp_collections.htm
- <https://github.com/danunora/unedl/tree/master/Command/Collections>

Unidad III, Microsoft Visual C# Avanzado

3.1 Colecciones genéricas (Generic Collection)

- Este tipo de colecciones pueden almacenar cualquier tipo de elementos.
- La limitante es que se debe utilizar “casting” para poder recorrer el contenido, de lo contrario se generará un excepción.
- Se impacta el rendimiento en su uso por el *boxing/unboxing*.
- El namespace de ***System.Collections.Generic*** contiene las siguientes colecciones: List<T>, Dictionary<TKey,TValue>, SortedList<TKey,TValue>, HashSet<T>, Queue<T>, Stack<T>

Ejemplo:

<http://www.tutorialsteacher.com/csharp/csharp-generic-collections>

Unidad III, Microsoft Visual C# Avanzado

3.1 Colecciones genéricas (Generic Collection)

Generic Collections	Description
List<T>	Generic List<T> contains elements of specified type. It grows automatically as you add elements in it.
Dictionary<TKey,TValue>	Dictionary<TKey,TValue> contains key-value pairs.
SortedList<TKey,TValue>	SortedList stores key and value pairs. It automatically adds the elements in ascending order of key by default.
HashSet<T>	HashSet<T> contains non-duplicate elements. It eliminates duplicate elements.
Queue<T>	Queue<T> stores the values in FIFO style (First In First Out). It keeps the order in which the values were added. It provides an Enqueue() method to add values and a Dequeue() method to retrieve values from the collection.
Stack<T>	Stack<T> stores the values as LIFO (Last In First Out). It provides a Push() method to add a value and Pop() & Peek() methods to retrieve values.

Ejemplo:

<http://www.tutorialsteacher.com/csharp/csharp-generic-collections>

Unidad III, Microsoft Visual C# Avanzado

3.1 Colecciones (list, hash, sorted list, stack, queue, bit array)

Sorting Algorithms	Descripción
Quick Sort	El algoritmo inicia partiendo de un pivote particionando la lista de elementos en dos. Los elementos mayores al pivote van a la derecha y los menores al pivote a la izquierda. Se hace el mismo procedimiento en la parte derecha y en la parte izquierda, se puede utilizar recursividad. Su complejidad es de $O(N^2)$ en el peor caso.
Bubble Sort	Algoritmo basado en la comparación de pares, en el que después de la comparación se decide si se intercambian las posiciones de la dupla. Su complejidad es de $O(N^2)$ en el peor caso.
Merge Sort	El método Quicksort divide la estructura en dos y ordena cada mitad recursivamente. El caso del MergeSort es el opuesto, es decir, en éste método se unen dos estructuras ordenadas para formar una sola ordenada correctamente. Su complejidad es $O(n \log n)$ en el peor caso

Unidad III, Microsoft Visual C# Avanzado

3.1 Colecciones (list, hash, sorted list, stack, queue, bit array)

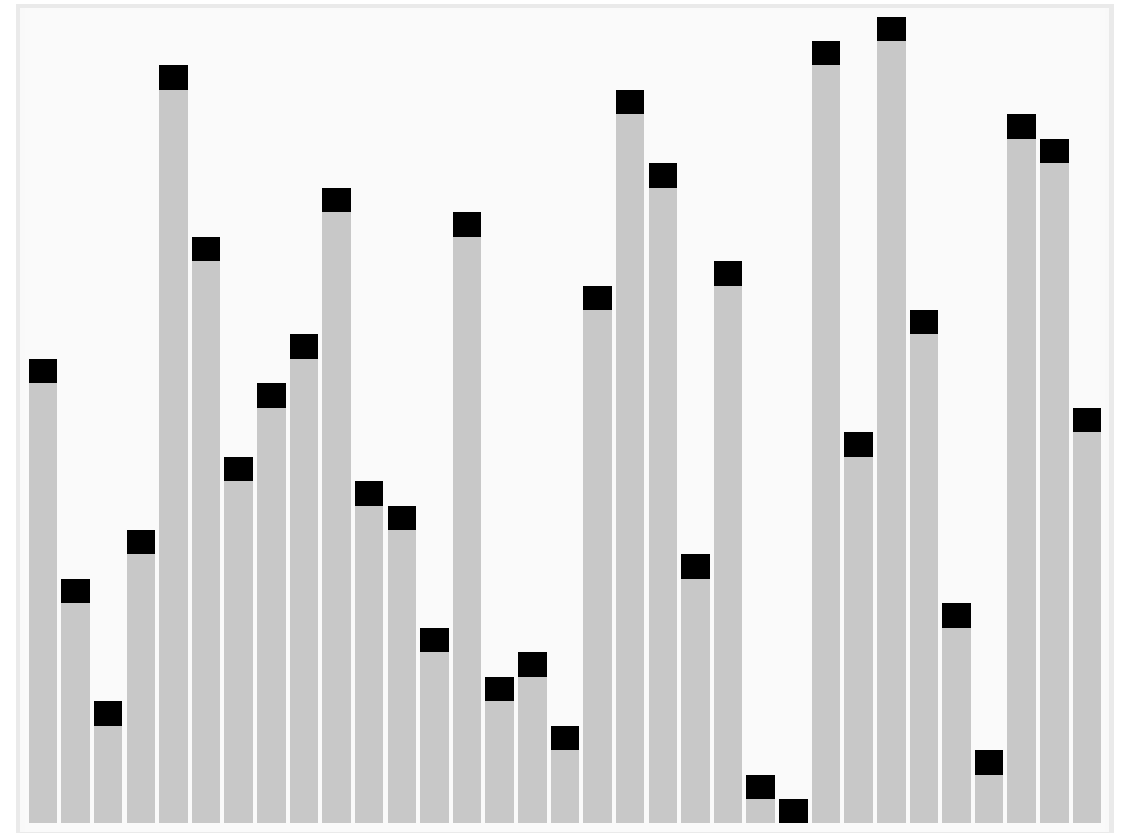
Sorting Algorithms	Descripción
Heap Sort	<p>Se convierte un arreglo en una cola de prioridad (heap), se construye un arreglo ordenado de atrás hacia adelante (mayor a menor) repitiendo los siguientes pasos:</p> <ol style="list-style-type: none">1) sacar el valor máximo en el heap (el de la posición 1)2) poner ese valor en el arreglo ordenado3) reconstruir el heap con un elemento menos4) utilizar el mismo arreglo para el heap y el arreglo ordenado <p>Su complejidad es $O(n \log n)$ en el peor caso</p>
Shell Sort	<p>Denominado así por su desarrollador Donald Shell (1959), ordena una estructura de una manera similar a la del Bubble Sort, sin embargo no ordena elementos adyacentes sino que utiliza una segmentación entre los datos</p>

Unidad III, Microsoft Visual C# Avanzado

3.1 Colecciones (list, hash, sorted list, stack, queue, bit array)

Quick Sort,

- El algoritmo inicia partiendo de un pivote particionando la lista de elementos en dos. Los elementos mayores al pivote van a la derecha y los menores al pivote a la izquierda.
- Se hace el mismo procedimiento en la parte derecha y en la parte izquierda, se puede utilizar recursividad. Su complejidad es de $O(N^2)$ en el peor caso.
- Ejemplo:
- https://www.tutorialspoint.com/data_structures_algorithms/quick_sort_algorithm.htm



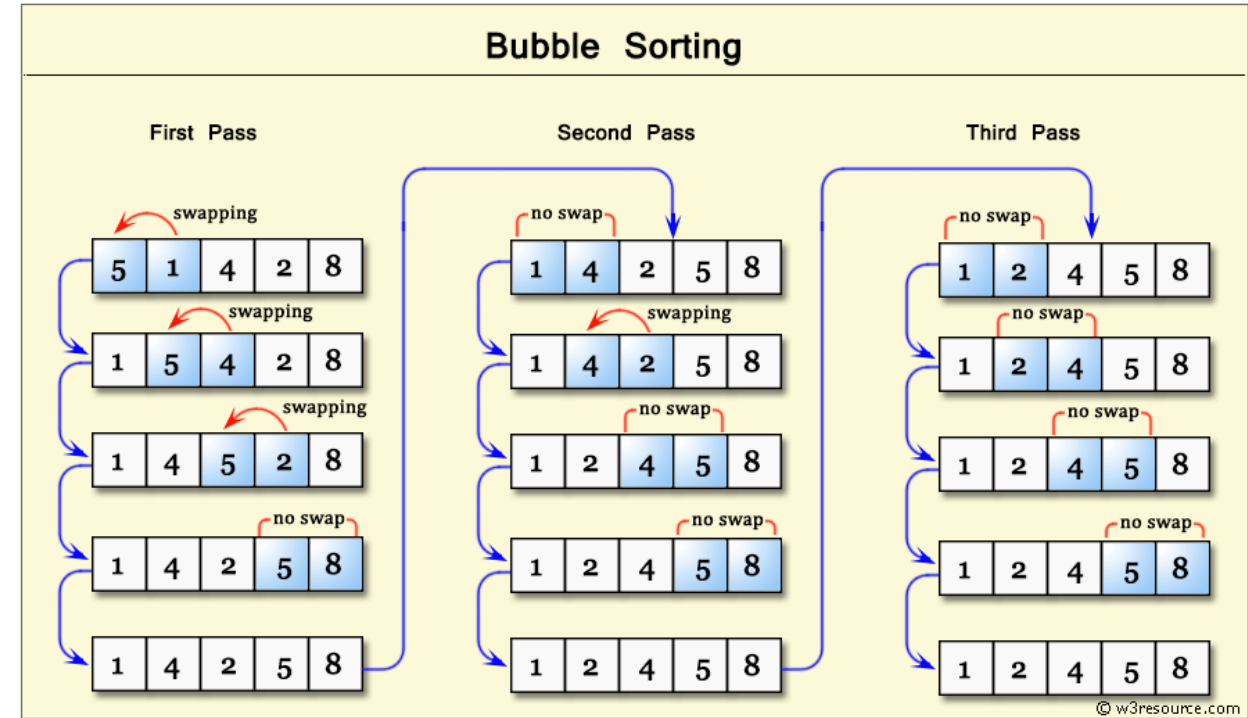
Sorting animation (Source: <http://en.wikipedia.org/wiki/Quicksort>)

Unidad III, Microsoft Visual C# Avanzado

3.1 Colecciones (list, hash, sorted list, stack, queue, bit array)

Bubble Sort,

- Algoritmo basado en la comparación de pares, en el que después de la comparación se decide si se intercambian las posiciones de la dupla.
- Ejemplo:
- <https://www.c-sharpcorner.com/blogs/bubble-sort-algorithm-in-c-sharp>



Sorting animation (Source: <https://www.w3resource.com/csharp-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-3.php>)

Unidad III, Microsoft Visual C# Avanzado

3.1 Colecciones (list, hash, sorted list, stack, queue, bit array)

Merge Sort,

- Se unen dos estructuras ordenadas para formar una sola ordenada correctamente.

- Ejemplo:

https://www.tutorialspoint.com/data_structures_algorithms/merge_sort_algorithm.htm

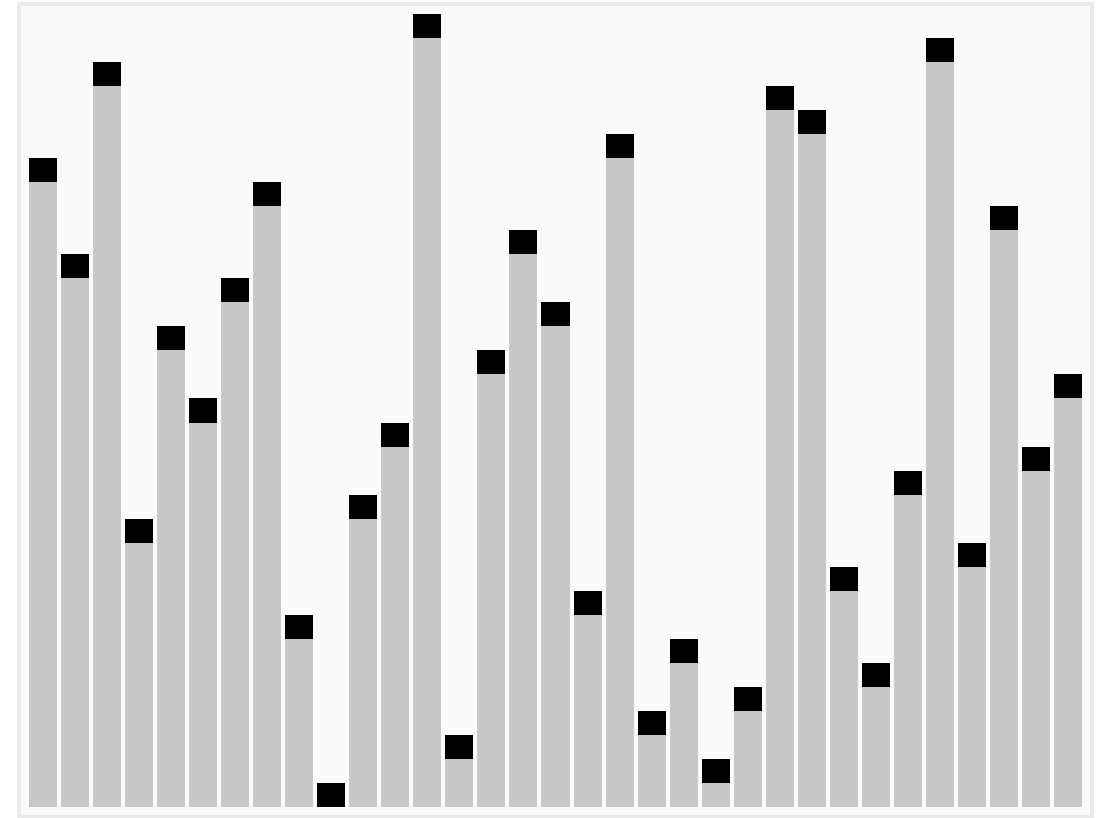
6 5 3 1 8 7 2 4

Unidad III, Microsoft Visual C# Avanzado

3.1 Colecciones (list, hash, sorted list, stack, queue, bit array)

Heap Sort,

- Cola de Prioridad
- Ejemplo:
- <https://www.tutorialspoint.com/Heap-Sort>

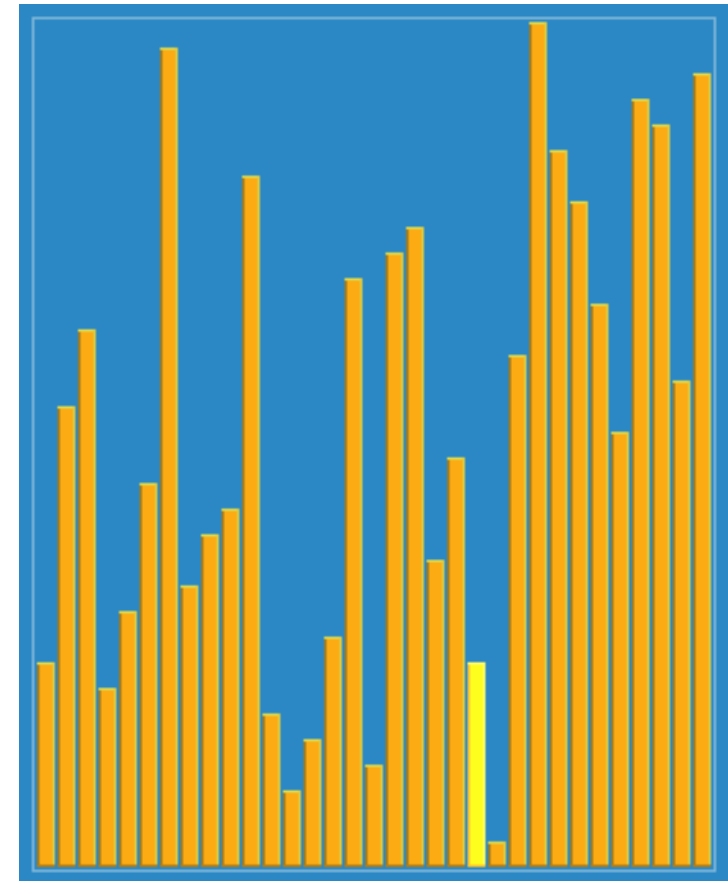


Unidad III, Microsoft Visual C# Avanzado

3.1 Colecciones (list, hash, sorted list, stack, queue, bit array)

Shell Sort,

- Ordena una estructura de una manera similar a la del Bubble Sort, sin embargo no ordena elementos adyacentes sino que utiliza una segmentación entre los datos
- Ejemplo:
- https://www.tutorialspoint.com/data_structures_algorithms/shell_sort_algorithm.htm



Sorting animation (Source: <https://en.wikipedia.org/wiki/Shellsort>)

Unidad III, Microsoft Visual C# Avanzado

TAREA # 6 algoritmos de ordenamiento C#:

1. Hacer una forma que pida un conjunto de números enteros, agregar botones que permitan implementar los siguientes algoritmos de ordenamiento: quick sort, bubble sort, merge sort y heap sort, agregar un campo que despliegue el resultado.
2. Subir el programa a github o bitbucket y enviar por email (daniel_nuno@hotmail.com) la liga al mismo.



Unidad III, Microsoft Visual C# Avanzado

3.2 Control de excepciones.

Una excepción es un evento inesperado que aparece durante la ejecución de un programa.

Una excepción es la respuesta a una circunstancia excepcional durante la ejecución del programa, que puede ser al tratar de acceder a un elemento fuera de límite, la división entre cero, etc.

Las excepciones se manejan utilizando las siguientes instrucciones:

- “try”, “catch”, “finally” y “throw”.

Unidad III, Microsoft Visual C# Avanzado

3.2 Control de excepciones.

- “**try**”, se usa para identifica un bloque de código por el cual alguna excepción se podría llevar a cabo.
- “**catch**”, acompaña al “try” y sirve para atrapar a una o mas excepciones que podrían generarse en el bloque del “try”.
- “**finally**”, se utiliza para ejecutar una acción independiente si hubo o no una excepción.
- “**throw**”, se utiliza para lanzar una excepción cuando se encuentra algún problema.

Unidad III, Microsoft Visual C# Avanzado

3.2 Control de excepciones (Sintaxis)

```
try {  
    // statements causing exception  
} catch( ExceptionName e1 ) {  
    // error handling code  
} catch( ExceptionName e2 ) {  
    // error handling code  
} catch( ExceptionName eN ) {  
    // error handling code  
} finally {  
    // statements to be executed  
}
```

Unidad III, Microsoft Visual C# Avanzado

3.2 Control de excepciones.

En C# las excepciones se representan mediante clases que son derivadas de la clase **System.Exception** y se derivan en:

- **System.ApplicacionException**, son excepciones generadas por la aplicación
- **System.SystemException**, son excepciones generados por el sistema.

Ejemplo:

<https://github.com/danunora/unedl/tree/master/Command/Exceptions>

https://www.tutorialspoint.com/csharp/csharp_exception_handling.htm

Unidad III, Microsoft Visual C# Avanzado

3.2 Control de excepciones.

Algunas de las excepciones más comunes derivadas de **System.SystemException** son:

Excepción	Descripción
System.IO.IOException	Errores de entrada y salida
System.DivideByZeroException	División ente cero
System.OutOfMemoryException	Memoria libre insuficiente
System.StackOverflowException	Desbordamiento de la pila
System.NullReferenceException	Refencia hacia un objeto nulo

Ejemplos:

<https://github.com/danunora/unedi/tree/master/Command/Exceptions>

https://www.tutorialspoint.com/csharp/csharp_exception_handling.htm

Unidad III, Microsoft Visual C# Avanzado

Ejercicio # 6 en clase:

Control de Excepciones

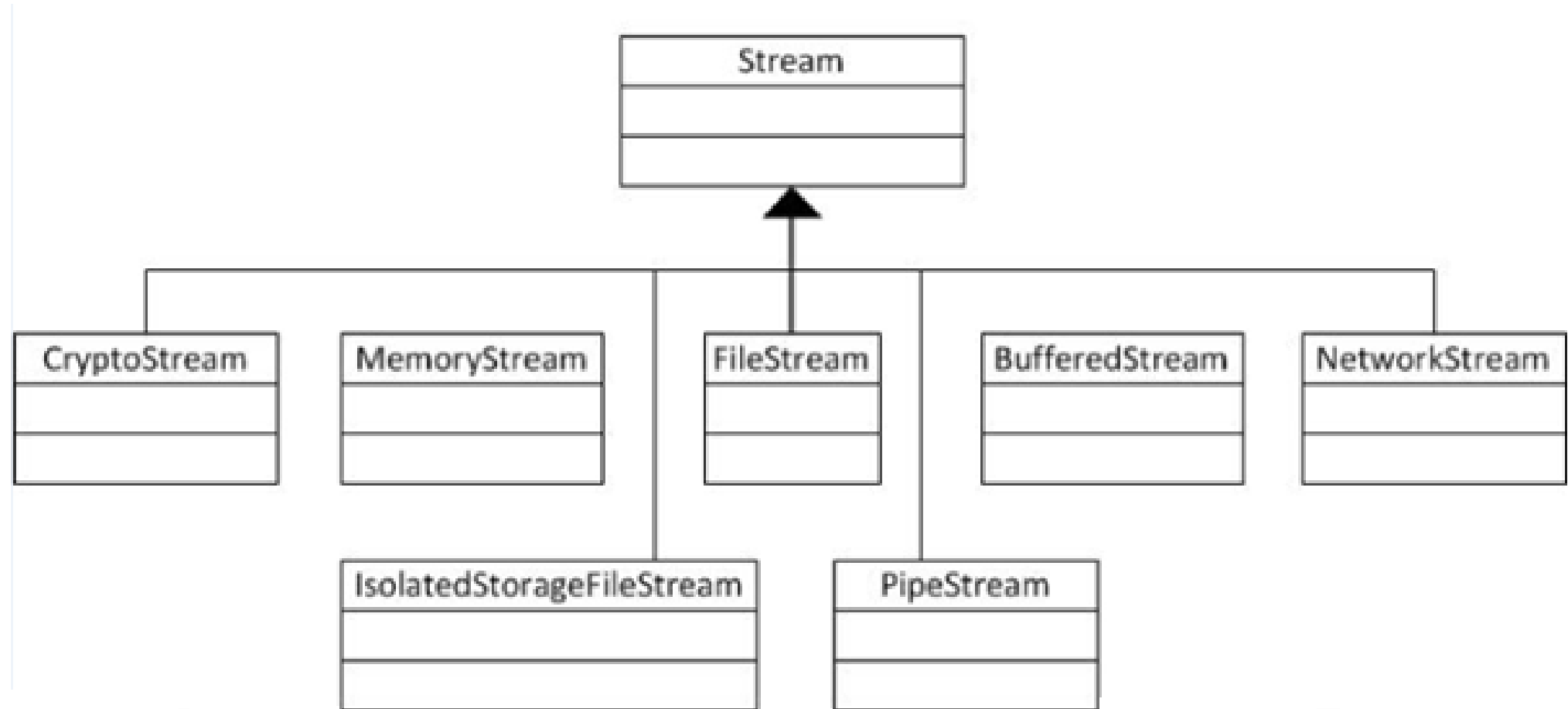
1. En base a la página:
 1. <https://docs.microsoft.com/en-us/dotnet/api/system.exception>
2. Realizar el ejercicio de implementar una excepción personalizada.
3. Subir el programa a github o bitbucket y enviar por email (daniel_nuno@hotmail.com) la liga al mismo.

Unidad III, Microsoft Visual C# Avanzado

3.3 Entrada y salida.

- C# incluye clases para leer de diferentes entradas y escribir en diferentes salidas.
- **Stream: System.IO.Stream** es una clase abstracta que provee diferentes métodos para transferencia de bytes.
- Las siguientes clases son derivadas de la clase abstracta System.IO.Stream:
 - FileStream
 - MemoryStream
 - BufferedStream
 - PipeStream
 - CryptoStream

Unidad III, Microsoft Visual C# Avanzado



Unidad III, Microsoft Visual C# Avanzado

3.3 Entrada y salida.

- **FileStream**, lee y escribe bytes desde y hacia un archivo.
- **MemoryStream**, lee y escribe bytes desde y hacia memoria.
- **BufferedStream**, lee y escribe bytes desde y hacia otros Streams para mejorar el rendimiento de las operaciones de entrada y salida.
- **PipeStream**, lee y escribe bytes desde y hacia diferentes procesos.
- **CryptoStream**, método para relacionar los streams y transformarlos en criptografía.

Unidad III, Microsoft Visual C# Avanzado

3.3 Entrada y salida.

FileStream,

Clase que permite leer y escribir bytes desde y hacia un archivo. Se deriva de la clase abstracta Stream.

Ejemplo:

<https://github.com/danunora/unedl/tree/master/Command/FileIO>

Unidad III, Microsoft Visual C# Avanzado

3.3 Entrada y salida.

MemoryStream,

Clase que sirve para leer y escribir bytes desde y hacia memoria. Se deriva de la clase abstracta Stream.

Ejemplo:

<https://www.dotnetperls.com/memorystream>

Unidad III, Microsoft Visual C# Avanzado

3.3 Entrada y salida.

StreamReader y **StreamWriter** son clases de apoyo que permiten la lectura y escritura convirtiendo bytes en caracteres y viceversa.

Se pueden utilizar para diferentes Streams como **FileStream**, **MemoryStream**, etc.



Ejemplo:

<https://www.guru99.com/c-sharp-stream.html>

Unidad III, Microsoft Visual C# Avanzado

3.3 Entrada y salida.

BinaryReader y **BinaryWriter** son clases de apoyo que permiten la lectura y escritura de tipos de dato primitivos en bytes y viceversa.

Ejemplos:

https://www.tutorialspoint.com/csharp/csharp_file_io.htm

https://www.tutorialspoint.com/csharp/csharp_binary_files.htm

<https://www.codeproject.com/Articles/16011/PipeStream-a-Memory-Efficient-and-Thread-Safe-Stre>

Unidad III, Microsoft Visual C# Avanzado

3.4 Trabajando con “DateTime” y “TimeSpan”

Existen dos estructuras en el “namespace” de “System” que nos permiten trabajar con fechas y tiempo, y una vez que han sido instanciadas, no pueden cambiar su valor.

DateTime, es una estructura que representa un instante en el tiempo, se expresa en fecha y hora de un día.

TimeSpan, es una estructura que representa un intervalo de tiempo.

Ejemplos:

- <https://docs.microsoft.com/en-us/dotnet/api/system.datetime?view=netframework-4.7.2>
- <https://docs.microsoft.com/en-us/dotnet/api/system.timespan?view=netframework-4.7.2>
- https://github.com/danunora/unedi/tree/master/Command/DateTime_TimeSpan

Unidad III, Microsoft Visual C# Avanzado

Ejercicio # 7 en clase:

Reloj Checador

1. En base a las páginas:

1. https://www.tutorialspoint.com/csharp/csharp_file_io.htm
2. <https://github.com/danunora/unedi/tree/master/Command/DateTime TimeSpan>

2. Realizar el ejercicio de implementar un Reloj Checador.

1. Registrar entradas y salidas de varias personas en un archivo de texto
2. Formato del registro: <persona><entrada/salida><fecha><hora>
3. Windows Form
 1. Forma de Registro
 2. Forma de Búsqueda por persona de entradas y salidas, debe listar el contenido del archivo de txt.

3. Subir el programa a github o bitbucket y enviar por email (daniel_nuno@hotmail.com) la liga al mismo.

Unidad III, Microsoft Visual C# Avanzado

3.5 Espacios de nombres

- Un “namespace” fue designado para separar un grupo de nombres de las clases de otro.
- Las clases declaradas en un “namespace” no hacen conflicto con las clases de otro “namespace” que contengan el mismo nombre.
- Un namespace tiene la siguiente sintaxis:

```
namespace namespace_name {  
    // code declarations  
}
```

Unidad III, Microsoft Visual C# Avanzado

3.5 Espacios de nombres

```
using System;
```

```
namespace first_space {  
    class namespace_cl {  
        public void func() {  
            Console.WriteLine("Inside first_space");  
        }  
    }  
}
```

```
namespace second_space {  
    class namespace_cl {  
        public void func() {  
            Console.WriteLine("Inside second_space");  
        }  
    }  
}
```

Unidad III, Microsoft Visual C# Avanzado

3.5 Espacios de nombres

Se selecciona que espacio de nombres se esta utilizando con la palabra “**using**” seguida del nombre del espacio de nombres.

Los espacios de nombres también se pueden anidar dentro de otro espacio de nombres.

Ejemplo:

<https://github.com/danunora/unedl/tree/master/Command/Namespaces>

Unidad III, Microsoft Visual C# Avanzado

3.6 Preprocesador y ensamblados.

- Las **directivas para el preprocesador** le dan instrucciones al compilador para que procese la información antes de iniciar con la propia compilación.
- Las directivas inician con “#” y no terminan con “;”.
- Las directivas deben ser las únicas instrucciones en la línea.
- El compilador de C# no contiene un preprocesador, pero actúa como si lo tuviera.

Unidad III, Microsoft Visual C# Avanzado

3.6 Preprocesador y ensamblados.

Todas las directivas son:

#define	#undef	#if	#else
#elif	#endif	#line	#error
#warning	#region	#endregion	

Unidad III, Microsoft Visual C# Avanzado

3.6 Preprocesador y ensamblados.

- La directiva **#define**, se utiliza para definir constantes simbólicas.
- **#define** permite definir un símbolo de tal manera que si el símbolo se utiliza en una condicional, la expresión se evalúa como verdadera.
- Se utiliza **#define** al inicio de un archivo fuente, antes del primer símbolo y su alcance esta limitado al mismo archivo.

```
#define PI
#if (PI)
Console.WriteLine("PI is defined");
#else
Console.WriteLine("PI is not defined");
#endif
```

Unidad III, Microsoft Visual C# Avanzado

3.6 Preprocesador y ensamblados.

- Las **directivas condicionales** son:
 - `#if`, evalúa símbolo o símbolos a verdadero
 - `#else`, evalúa el valor contrario de `#if`
 - `#elif`, es una nueva evaluación en la misma estructura
 - `#endif`, especifica el fin de un condicional
 - Se pueden utilizar los siguientes operadores:
 - `==` (equality), `!=` (inequality), `&&` (and), `||` (or)

Ejemplo:

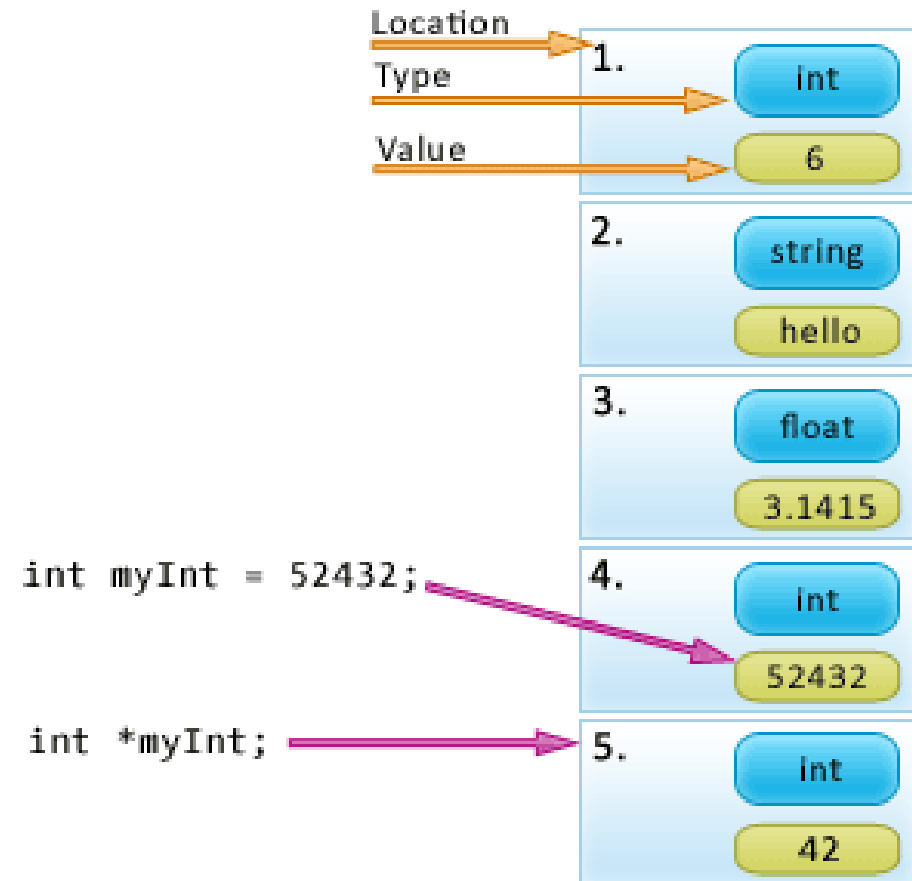
<https://github.com/danunora/unedi/tree/master/Command/PreProcessor>

Unidad III, Microsoft Visual C# Avanzado

3.7 Código no seguro y apuntadores en C#.

- Un apuntador es una variable cuyo valor contiene la dirección de otra variable, por ejemplo un la dirección de una localidad de memoria.
- La declaración de un apuntador es como sigue: `type *var-name;`

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```



Unidad III, Microsoft Visual C# Avanzado

3.7 Código no seguro y apuntadores en C#.

- A diferencia de los tipos de referencia, los apuntadores no son rastreados por los mecanismos del recolector de basura.
- Por la misma razón, a los apuntadores no se les permite apuntar a un tipo de referencia o a alguna estructura que contenga tipos de referencia.
- Los apuntadores únicamente pueden ser usados para apuntar a los tipos de datos básicos.

Unidad III, Microsoft Visual C# Avanzado

3.7 Código no seguro y apuntadores en C#.

Código no seguro

El código de C# se puede ejecutar ya sea en el contexto seguro o en el contexto inseguro.

El código ejecutado fuera del contexto seguro, se ejecuta fuera del control del recolector de basura(*garbage collector*).

Cualquier código que involucre el uso de apuntadores requiere del uso del contexto inseguro.

Unidad III, Microsoft Visual C# Avanzado

3.7 Código no seguro y apuntadores en C#.

Código no seguro

Se puede invocar el modo inseguro desde dos maneras diferentes:

1. Utilizando un modificador '**unsafe**' para un método, propiedad, y un constructor.
2. Utilizando la palabra '**unsafe**' para definir un grupo de código.
3. Se debe habilitar en el compilador el uso de unsafe, de otra manera se generara un error en la compilación: [Compiler Error CS0227](#)
 1. <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/compiler-options/unsafe-compiler-option>

Unidad III, Microsoft Visual C# Avanzado

3.7 Código no seguro y apuntadores en C#.

Cuando usar Código no seguro

1. En caso de usar apuntadores
2. Cuando se escribe código que interactúa con el sistema operativo
3. Cuando se requiere implementar un algoritmo donde el tiempo es crítico
4. Cuando se requiere acceder a segmentos de memoria.

Unidad III, Microsoft Visual C# Avanzado

3.7 Código no seguro y apuntadores en C#.

Ejemplos:

- <https://github.com/danunora/unedl/tree/master/Command/UnsafeCode> Pointers
- <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/unsafe-code-pointers/index>
- https://www.tutorialspoint.com/csharp/csharp_unsafe_codes.htm
- <https://www.c-sharpcorner.com/article/pointers-in-C-Sharp/>
- <https://www.c-sharpcorner.com/UploadFile/f0b2ed/understanding-unsafe-code-in-C-Sharp/>

Unidad III, Microsoft Visual C# Avanzado

3.8 Expresiones Lambda

- Una expresión lambda es una función anónima que puede ser usada por delegados o arboles de expresiones.
- Se crean funciones locales que pueden ser pasadas como argumentos o regresadas como valor de una llamada a una función.



Unidad IV, Exploración de la biblioteca C#



Unidad IV, Exploración de la biblioteca C#

4.1 Exploración del espacio de nombres System.

4.2 Uso de clases principales predefinidas.

4.3 Administración de memoria.

4.4 El recolector de basura de C#.

Unidad IV, Exploración de la biblioteca C#

4.1 Exploración del espacio de nombres **System**.

Contiene clases fundamentales y clases que sirven de base para definir tipos de datos y tipos de referencia, eventos, interfaces, atributos y el procesamiento de excepciones.

Referencia:

<https://docs.microsoft.com/en-us/dotnet/api/system>

Unidad IV, Exploración de la biblioteca C#

4.2 Uso de clases principales predefinidas.

System

System.Collections

System.Data

System.Drawing

System.IO

System.Text

System.Threading

System.Timers

System.Web

System.Web.Services

System.Windows.Forms

System.Xml

Referencia:

<https://msdn.microsoft.com/en-us/library/ms973806.aspx>

Unidad IV, Exploración de la biblioteca C#

TAREA # 7 Ver video y realizar un ensayo

1. Ver el siguiente video acerca de manejo de memoria y el recolector de basura, y realizar un ensayo:
2. <https://www.youtube.com/watch?v=9FEfy9y0fFQ>
3. Subir el programa/documento a github o bitbucket y enviar por email (daniel_nuno@hotmail.com) la liga al mismo.

Unidad IV, Exploración de la biblioteca C#

4.3 Administración de Memoria

.NET Runtime

- Maneja la ejecución de los programas
- Just-in-time compilation: Intermediate Language (IL) -> machine code
- Type safety
- Exception handling
- Security
- Thread Management
- Memory Management
- Garbage collection (GC)

Unidad IV, Exploración de la biblioteca C#

4.3 Administración de memoria.

- La combinación entre el manejo de memoria y el recolector de basura (*garbage collector*), permiten tener virtualmente memoria ilimitada para las aplicaciones.
- Funciona mediante el uso de una gran porción de memoria que .NET provee cuando la aplicación se inicia.
- Eventualmente .NET continua la gestión de esa memoria.
- Posteriormente, el recolector de basura frecuentemente reclama memoria sin uso y la hace accesible nuevamente.

Unidad IV, Exploración de la biblioteca C#

4.3 Administración de memoria.

- La memoria que fue alojada por .NET, se le llama “managed heap” y es utilizada para proveer memoria a los objetos al instanciarlos.
- Es muy rápida, pues el *runtime* no necesita ir con el sistema operativo, sino que únicamente utiliza un apuntador a esa sección de memoria.
- Se utiliza también memoria que no es gestionada (unmanaged) y es para procesos internos del *runtime*: .NET CLR, Dynamic libraries, Graphics buffers, etc.
- Dicha memoria no es gestionada por el recolector de basura.

Unidad IV, Exploración de la biblioteca C#

4.3 Administración de memoria.

- La memoria de los tipos primitivos (int, bool, struct, decimal, enum, float, byte, long, ...) no es alojada en el **heap**, sino que se utiliza el **stack**.
- El ***stack*** no es manejado por el **Garbage Collector**.
- Para los tipos de referencia (clases), se utiliza el **heap**, cada vez que se usa “new” para instanciar y cuando se declaran cadenas vacías (“”).

Unidad IV, Exploración de la biblioteca C#

4.4 Recolector de Basura (Garbage collector)

- Su función principal es liberar memoria que no esta en uso.
- GC libera objetos que no ya no son usados mediante examinar su origen (raíz).
- GC arma un grafo con todos los objetos que son accesibles a dicha raíz, entonces si el objeto en cuestión no es accesible, remueve el objeto, libera la memoria o la agrega a la cola de liberación, y posteriormente compacta la porción de memoria alojada inicialmente (heap) evitando la fragmentación.
- Toma tiempo el buscar esos objetos (scan) por lo que el GC implementa un método para hacerlo.

Unidad IV, Exploración de la biblioteca C#

4.4 Recolector de Basura (Garbage collector)

- Generaciones

Generation 0	Generation 1	Generation 2
Objetos de poca vida, ejemplo: variables locales.	Objetos intermedios	Objetos que son mas utilizados o tienen más tiempo de vida, ejemplo: La ventana principal de la aplicación.

En la generación 0 se recolecta la basura mas frecuentemente, por lo que aquellos objetos que todavía se encuentren en uso en cada pasada, serán movidos hacia una generación más alta. Si el objeto ya esta en la generación 2, simplemente permanecerá ahí.

Si el objeto ya no se encuentra disponible, el GC simplemente liberará la memoria que usaba.

Unidad IV, Exploración de la biblioteca C#

4.4 Recolector de Basura (Garbage collector)

Large Object Heap (LOH)

- Es un segmento especial para objetos grandes (>85k)
- Se recolectan únicamente con una recolección de basura total, puesto que al ser mas grandes tienen mas propiedades y es mas complejo estarlos revisando.
- No serán compactados por defecto
- La fragmentacion podría generar una excepción de **OutOfMemoryException**

Unidad IV, Exploración de la biblioteca C#

4.4 Recolector de Basura (Garbage collector)

La ejecución del recolector de basura es variable y las siguientes condiciones pueden influir en ella:

- Condición del sistema de ya no tener memoria, no se puede alojar.
- Después de una asignación de memoria considerable
- Falla al alojar algunos recursos internos.
- El uso del [Profiler](#)
- Se hace de manera forzada al invocar métodos del **System.GC**
- La aplicación se va al segundo plano.

Unidad IV, Exploración de la biblioteca C#

4.4 Recolector de Basura (Garbage collector)

Garbage Collector Source Code

<https://raw.githubusercontent.com/dotnet/coreclr/master/src/gc/gc.cpp>

Unidad IV, Exploración de la biblioteca C#

4.4 Recolector de Basura (Garbage collector)

- Se ejecuta frecuentemente en gen0 pues hay menos referencias a variables, objetos, etc.
- En generaciones posteriores (gen1 y gen2) hay mas referencias a objetos, y son mas difíciles de limpiar, por lo que el recolector de basura tarda mas en realizar su tarea.
- Cuando se ejecuta el recolector de basura, se pausa la aplicación.
- El GC se ejecuta en un diferente *thread* y en segundo plano por defecto.

Unidad IV, Exploración de la biblioteca C#

4.4 Recolector de Basura (Garbage collector)

Recomendaciones:

- Uso de la interfaz [IDisposable](#) / **using** statement
- Utilizar destructores ([finalizers](#)), para mover los objetos a la cola de objetos para destruir.
- Utilizar referencias débiles para permitir al GC recolectarlas sin necesidad de revisarlas.

Ejemplo:

<https://github.com/maartenba/memory-demos>

Unidad IV, Exploración de la biblioteca C#

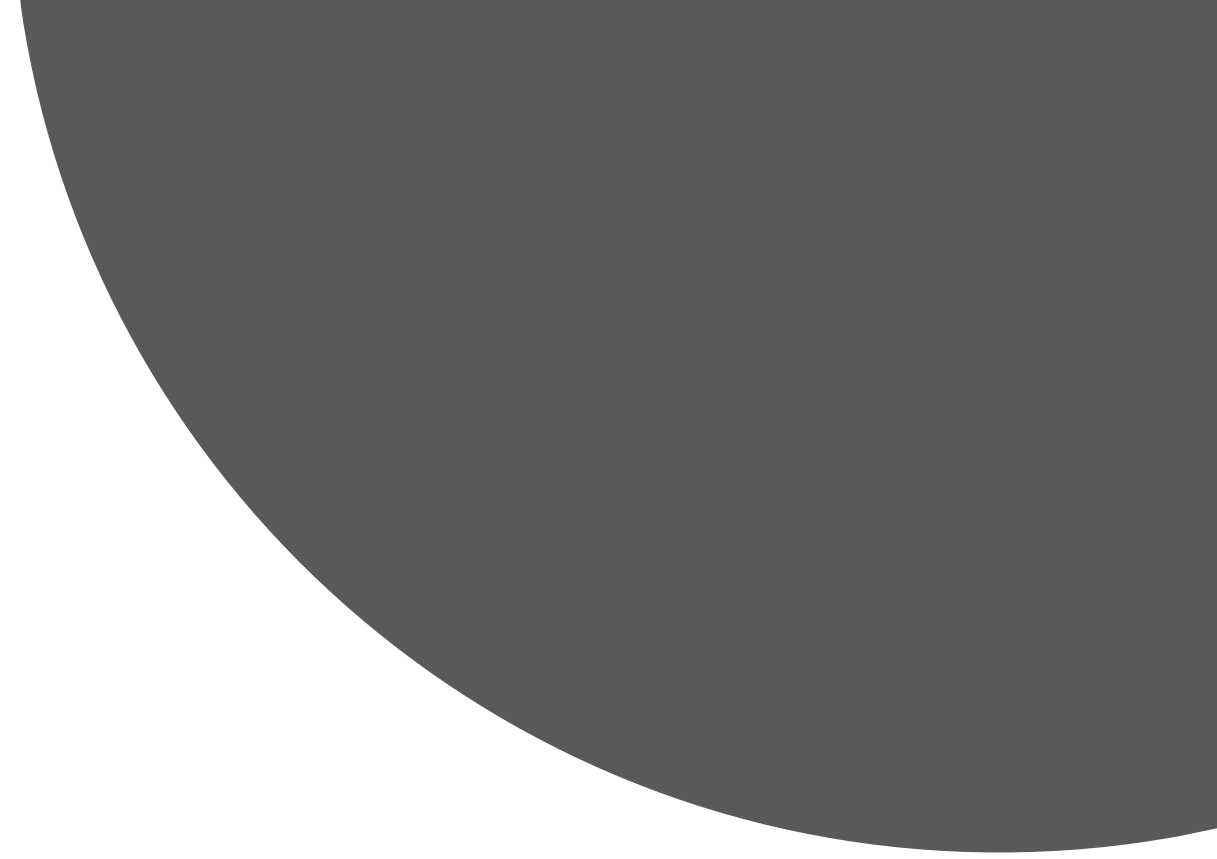
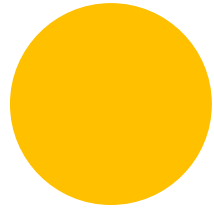
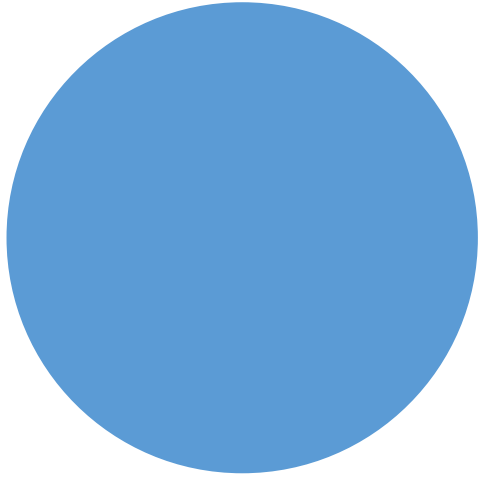
Referencias:

- ReSharper Heap Allocations Viewer plugin
- <https://blog.jetbrains.com/dotnet/2014/06/06/heap-allocations-viewer-plugin/>
- Roslyn's Heap Allocation Analyzer
- <https://github.com/Microsoft/RoslynClrHeapAllocationAnalyzer>
- Roslyn C# Heap Allocation Analyzer video
- <https://www.youtube.com/watch?v=Tw-wgT-cXYU&hd=1>

Unidad IV, Exploración de la biblioteca C#

TAREA # 8 Investigación acerca de GC en Java

1. Ver el siguiente video acerca del recolector de basura, y realizar un ensayo:
2. https://www.youtube.com/watch?v=Uj1_4shgXpk
3. Subir el programa/documento a github o bitbucket y enviar por email (daniel_nuno@hotmail.com) la liga al mismo.



Unidad V, Programación Multiproceso



Unidad V, Programación Multiproceso

- 5.1 Fundamentos del multiproceso.
- 5.2 La clase Thread.
- 5.3 Propiedades de los procesos.
- 5.4 Comunicación entre procesos.
- 5.5 Suspensión, reanudación y detención de subprocesos.
- 5.6 Tareas independientes.

Unidad V, Programación Multiproceso

5.1 Fundamentos de Multiproceso

- Un *thread* esta definido como un medio para ejecutar un programa.
- Cada *thread* define un flujo de control único.
- Ideal para aplicaciones donde se involucren operaciones complicadas y que consumen mucho tiempo.
- Los *threas* son procesos ligeros que nos ayudan a dividir una sola tarea en varias.

Unidad V, Programación Multiproceso

Ciclo de Vida de un *Thread*

El ciclo de vida de un *thread* inicia cuando un objeto es creado a partir de la clase **System.Threading.Thread** y termina, cuando el *thread* es terminado o completa su ejecución.

Unstarted State	Una instancia del thread se crea pero no ha sido ejecutado
Ready State	Cuando el thread esta listo para su ejecución y en espera de un ciclo de CPU
Not Runnable State	Un thread no esta siendo ejecutado por cualquiera de las siguiente razones: <ul style="list-style-type: none">• Se ha llamado al método “Sleep”• Se ha llamado al método “Wait”• Esta bloqueado en espera de operaciones de I/O
Dead State	El thread a concluido su trabajo o ha sido abortado

Unidad V, Programación Multiproceso

5.2 La clase Thread

En C#, la clase **System.Threading.Thread** se utiliza para trabajar con threads.

Nos permite crear y acceder a ***threads*** individualmente, en una aplicación ***multithreaded***.

El primer proceso en ser ejecutado se conoce como ***main thread***

Se puede acceder a un *thread* utilizando la propiedad **CurrentThread**

Ejemplo:

https://www.tutorialspoint.com/csharp/csharp_multithreading.htm

Unidad V, Programación Multiproceso

¿Cómo se utilizan los ***Threads*** in .NET?

A partir de .NET Framework 4, la recomendación para utilizar Threads es a través del uso de:

- [Task Parallel Library \(TPL\)](#)
- [Parallel LINQ \(PLINQ\)](#)

Unidad V, Programación Multiproceso

Microsoft C# parallelism API

- Task Parallel Library (TPL)

Conjunto de tipos públicos y APIs que forman parte de los espacios de nombres (namespaces) usados para trabajar con *threads*.

Task is a TPL's unit of work

Su propósito es simplificar el trabajo del desarrollador para agregar paralelismo y concurrencia a sus aplicaciones.

- [System.Threading](#)
- [System.Threading.Tasks](#)

Unidad V, Programación Multiproceso

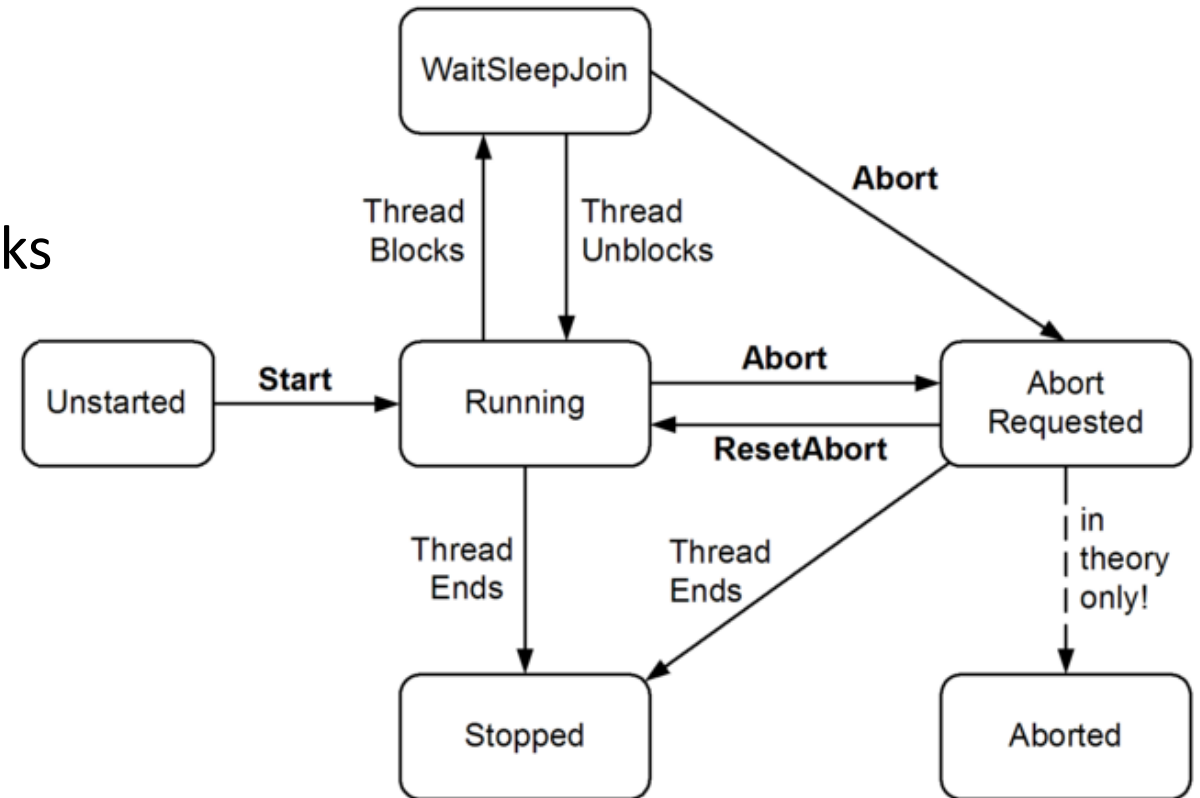
Microsoft C# parallelism API

- Parallel LINQ (PLINQ) es una implementación de los objetos LINQ(Language-Integrated Query, parallelized)
- Referencia:
- <https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/parallel-linq-plinq>

Unidad V, Programación Multiproceso

Microsoft C# parallelism API

- Task Parallel Programming
 - Creating and Starting Tasks
 - Cancelling Tasks
 - Waiting for Time to Pass or for Tasks
 - Exception Handling



Unidad V, Programación Multiproceso

Microsoft C# parallelism API

- Creating and Starting Tasks
 - Method1, Task is getting created and started
 - `Task.Factory.StartNew();`
 - Method2, Task is getting created and needs to be started
 - `var t = new Task();`
 - `t.Start();`

Example:

<https://github.com/danunora/unedi/tree/master/Command/TaskExampleStarting>

Unidad V, Programación Multiproceso

Microsoft C# parallelism API

- Cancelling Tasks
 - Method1
 - IsCancellationRequested?
 - break
 - Method2
 - IsCancellationRequested?
 - throw new OperationCanceledException()
 - Method3 (Recommended)
 - ThrowIfCancellationRequested()

Example:

<https://github.com/danunora/unedi/tree/master/Command/TaskExampleCancelling>

Unidad V, Programación Multiproceso

Microsoft C# parallelism API

- Waiting for Time to Pass
 - Thread.Sleep()
 - Pauses the thread of execution
 - Scheduler can get another task executed while sleep
 - [Thread.SpinWait\(\)](#)
 - SpinWait.SpinUnit(),
 - Pause the thread, but don't give up your place in the execution list
 - Wasting resources, but it's avoiding expensive context switches

Example:

<https://github.com/danunora/unedl/tree/master/Command/TaskExampleSpinWait>

<https://github.com/danunora/unedl/tree/master/Command/TaskExample2SpinWait>

<https://github.com/danunora/unedl/tree/master/Command/TaskExampleWaiting>

Unidad V, Programación Multiproceso

Microsoft C# parallelism API

- Task Status:

Task Value	Description
Created	Created but not started
WaitingForActivation	Waiting to be scheduled until some dependant operation has completed
WaitingToRun	Waiting for the scheduler
Running	Currently running.
WaitingForChildrenToComplete	Waiting for all attached children to complete
RanToCompletion	Final state. Success
Canceled	Final state. Was cancelled
Faulted	Final state. Unhandled exception.

- Source:

- <https://blogs.msdn.microsoft.com/pfxteam/2009/08/30/the-meaning-of-taskstatus/>

Unidad V, Programación Multiproceso

Microsoft C# parallelism API

- Exception Handling
 - Las tareas generan diferentes excepciones
 - Si no se manejan las excepciones en una tarea diferente podrían ser ignoradas
 - Se debe utilizar “**AggregateException**” para manejarlas y hacer un ciclo.

Referencia:


[https://msdn.microsoft.com/en-us/library/dd537614\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/dd537614(v=vs.110).aspx)

Example:

<https://github.com/danunora/unedi/tree/master/Command/TaskExampleExceptions>



Unidad VI, Creación de aplicaciones Windows



Unidad VI, Creación de aplicaciones Windows

6.1 Windows Forms App

6.2 Esqueletos de aplicaciones basadas en Windows Forms App.

6.3 La paleta de componentes.

6.4 Control de mensajes.

6.5 Principales componentes de una aplicación Windows.

Unidad VI, Creación de aplicaciones Windows

Proyecto Escolar

Examen de Certificación para Desarrolladores de Software

El proyecto consiste en realizar un examen a un usuario de sus conocimientos sobre programación visual y C#.

- Se tendrá un acervo de 100 preguntas mínimo, de las cuales se harán 20 preguntas por examen.
- Las preguntas serán de opción múltiple o de que el usuario seleccione la respuesta correcta.
- No habrá preguntas abiertas, ya que el software deberá calificar el examen al finalizarlo.
- El usuario deberá recibir su resultado por correo.

Unidad VI, Creación de aplicaciones Windows

Proyecto Escolar

Examen de Certificación para Desarrolladores de Software

Consideraciones:

- 1) El usuario deberá inscribirse utilizando su email, se deberá aceptar un email válido.
- 2) Las preguntas del examen se deberán sacar de un archivo de manera aleatoria.
- 3) Las preguntas del examen no podrán ser repetidas o haberse usado previamente.
- 4) Habrá 2 tipo de preguntas:
 - Opción Múltiple utilizando Check Boxes
 - Selección de opción, utilizando Radio Buttons
- 5) Para aceptar código, se deberá someter a "*code review*" incluyendo al profesor.

Unidad VI, Creación de aplicaciones Windows

Proyecto Escolar

Examen de Certificación para Desarrolladores de Software

Equipo 1, <https://bitbucket.org/unedl2018b4at1/equipo1/src/master/>

- 1) Kevin, zyotchatmajo@gmail.com
- 2) Santos, luisenrique.san.lan@gmail.com
- 3) Many, elmejorlui@gmail.com

Equipo 2, <https://bitbucket.org/unedl2018b4at2/equipo2/src/master/>

- 1) Miguel, mike.gm.117@gmail.com
- 2) Sergio, sherrera1695@gmail.com
- 3) Ivan, navi.248@hotmail.com
- 4) Octavio, parquer_17.camacho1297@hotmail.com

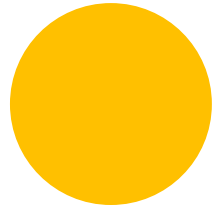
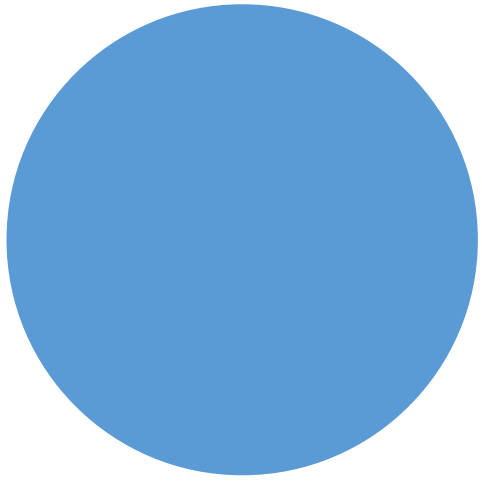
Unidad VI, Creación de aplicaciones Windows

6.1 Windows Forms App


Una forma es la unidad base de una aplicación.

Referencias:

- <https://docs.microsoft.com/en-us/dotnet/framework/winforms/index>
- <https://www.youtube.com/watch?v=AN79L5B7a58>
- <https://www.youtube.com/watch?v=B2DLZAXuFKY>
- <https://www.youtube.com/watch?v=zYmYx3pXNxU>
- https://www.youtube.com/watch?v=hD5_CfWkPDo



Unidad VII, El modelo de acceso a datos ADO.NET



Unidad VII, El modelo de acceso a datos ADO.NET

7.1 Características de ADO.NET.

7.2 Modo desconectado.

7.3 Modo conectado.

7.4 Controles data y controles enlazados
con Windows forms.

7.5 Elementos XML.



Anexos

Bibliografía

- Visual C# How to Program
 - by Harvey Deitel, Paul Deitel
 - Publisher: Pearson
 - Release Date: August 2016
 - ISBN: 9780134628820
- Microsoft Visual Studio 2015 Unleashed, Third Edition
 - by Mike Snell, Lars Powers
 - Publisher: Sams
 - Release Date: August 2015
 - ISBN: 9780134133164

Bibliografía

- C# for Java Developers
 - by Adam Freeman, Allen Jones
 - Publisher: Microsoft Press
 - Release Date: August 2002
 - ISBN: 9780735617797
- .NET Common Language Runtime Unleashed
 - by Kevin Burton
 - Publisher: Sams
 - Release Date: April 2002
 - ISBN: 0672321246

Referencias

- <https://docs.microsoft.com/en-us/dotnet/index>
- <https://visualstudio.microsoft.com/>
- <https://www.tutorialspoint.com/csharp/>
- <http://www.tutorialsteacher.com/csharp/csharp-tutorials>
- <http://www.albahari.com/threading/>
- Online Test
- https://www.tutorialspoint.com/csharp/csharp_online_test.htm

Java versus C#

Miscellaneous Language Differences

- The following list presents some simple language differences, just in case you are looking for a key feature:
- Java does not have foreach or goto.
- Java has no pointers.
- C# can drop down to unsafe mode.
- With C#, all types are derived from System.Object.
- That means that even primitive types have ToString, GetHashCode, GetType, and Equals methods available to them.
- In addition, collections can be made including primitive types and other types.
- Java does not have that option.

Java versus C#

Miscellaneous Language Differences

- Java has no operator overloading.
- Java has no preprocessor directives.
- Although the potential exists to have cross-platform IL with the CLR, it does not exist now, and Java has a clear advantage on this issue.
- Java does not have a struct or enum.
- C# adds a decimal type that Java does not have.
- C# has verbatim strings where escape characters are not interpreted.
- C# has in, out, and ref keywords to explicitly control how arguments are passed