Escuela de Sistemas y Tecnologías

Transparencias de **ANALISTA DE SISTEMAS**

Edición 2017 – Materia: **Diseño de Aplicaciones Web**

TEMA: Net Framework 2.0

Plantel y Contactos

- **Bedelía:**
  - Mail: bedeliasistemas@bios.edu.uy

- **Encargado de Sucursal:**
  - Pablo Castaño
  - Mail: pablocasta@bios.edu.uy

Recursos

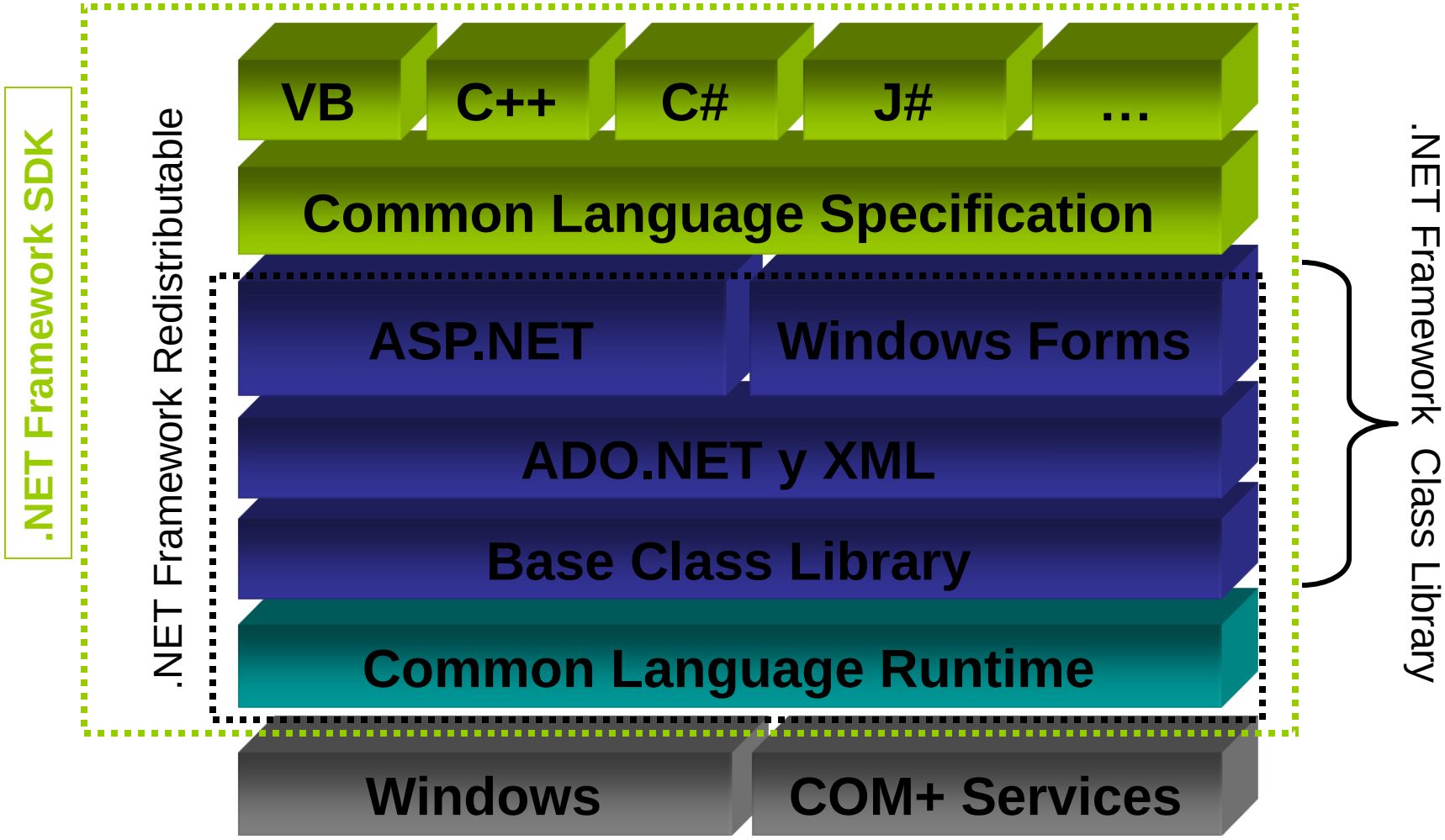
- **Recursos Imprescindibles:**
  - Sitio Web de material
    (comunicarse con Bedelía por usuario/contraseña).
  - Transparencias del Curso.
  - Contar con el software necesario

Agenda

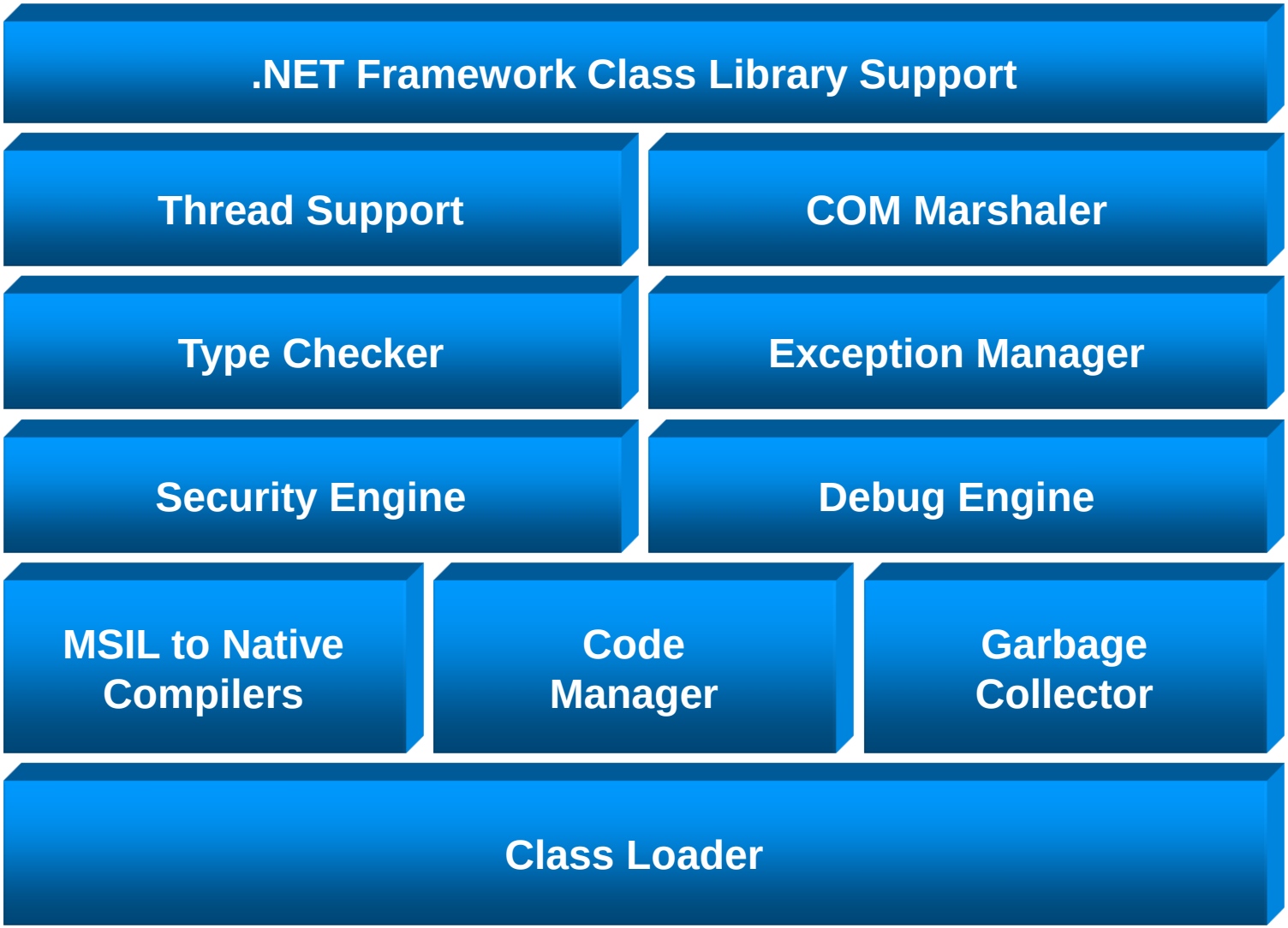
- Introducción
- Colecciones de Objetos
- Assembly
- Global Assembly Cache (GAC)
- Hilos de Ejecución (Threading)

Introducción

Arquitectura Net Framework



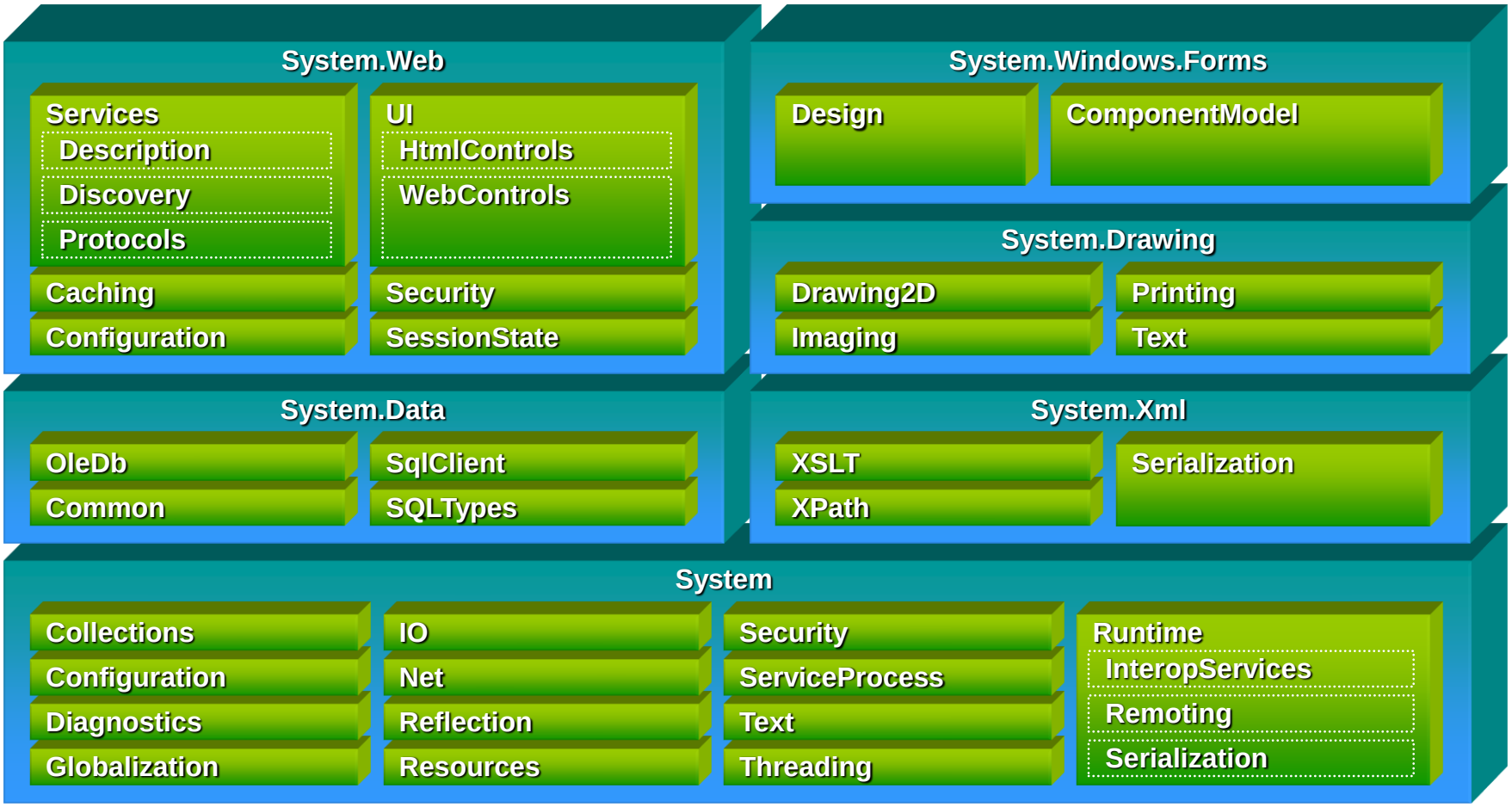
CLR - Arquitectura



CLR - Composición

- Common Languaje Runtime.
- Es el motor de ejecución (runtime) de .NET
- Características
  - ***Just-In-Time*** (Compilación JIT)
  - ***Garbage Collector*** (Gestión automática de memoria)
  - ***Excepciones*** (Gestión de errores consistente)
  - ***Assemblies*** (Ejecución basada en componentes)
  - Gestión de Seguridad
  - ***Multithreading***

BCL - Arquitectura



BCL - Composición

- Base Class Library.
- Conjunto de tipos básicos (clases, interfaces, etc.) predefinidos en el .NET Framework, independientes del lenguaje de desarrollo.
- Los tipos básicos están organizados en jerarquías lógicas de nombres, denominado ***NAMESPACE.***
- Es extensible y totalmente orientada a objetos.

CLS

- Common Language Specification.
- Especificación que estandariza las características soportadas por CLR .
- Contrato entre diseñadores de lenguajes de programación y autores de bibliotecas.
- Permite interoperabilidad entre lenguajes.
- Microsoft provee implementaciones de 4 lenguajes.
- Los tipos de aplicaciones .NET son independientes del lenguaje que se elija.

CTS

- Common Type System.
- Define un conjunto común de "tipos" de datos orientados a objetos.
- Todo lenguaje de programación .NET debe implementar los tipos definidos por el CTS.
- Todo tipo hereda directa o indirectamente del tipo System.Object.
- Define Tipos de ***valor*** y de ***referencia***.

Garbage Colector

- Cuando un objeto deja de ser útil (queda fuera del scope), debe ser liberado ese espacio.
- El ***garbage collector*** crea un gráfico de recolección (lista enlazada con jerarquía de instancias).
- El proceso de recolección consta de dos fases
  - Posiciones de heap con objetos fuera de scope o referencias null, marcadas como libre
  - Compactación de memoria

Generics (1)

- Son tipos parametrizados soportados por el CLR:
  - Un tipo parametrizado es aquel que puede definirse sin especificar los tipos de datos de sus parámetros en tiempo de compilación.
- Nos dan la posibilidad de declarar clases, estructuras, métodos e interfaces que actuarán uniformemente sobre valores cuyos tipos se desconocen a priori y son recién especificados al momento de su utilización.
- Provee el beneficio de código genérico:
  - Permite verificación en tiempo de ejecución de tipado fuerte.
  - Reduce la necesidad de conversiones explícitas de tipos.
  - Permite generar código limpio y mas seguro.

Generics (2)

```
public class ClaseGenerica<T>
{
    public T _atributo;
}
```

```
ClaseGenerica <string> _var = new ClaseGenerica<string>();
_var._atributo = “Un string";
_var._atributo = 2;   //Genera Error de Compilación
…
ClaseGenerica<int> _var 2 = new ClaseGenerica<int>();
_var 2._atributo = 2;
_var._atributo = “Un string"; ;   //Genera Error de Compilación
```

Colecciones de Objetos

System.Collection

- ***ArrayList***: vector cuyo número de elementos puede modificarse dinámicamente.
- ***HashTable***: El acceso a los valores del vector se realiza a través de una clave asociada a cada elemento.
- ***SortedList***: variación de un HashTable en la que los elementos se ordenan por la clave según van siendo agregados.
- ***Queue***: Útiles para almacenar objetos en el orden en el que fueron recibidos.
- ***Stack***: colección de objetos simple de la clase último en entrar, primero en salir

HashTable

```
using System;
using System.Collections;
public class SamplesHashtable
{
    public static void Main()
    {
        // Creates and initializes a new Hashtable.
        Hashtable myHT = new Hashtable();
        myHT.Add("First", "Hello");
        myHT.Add("Second", "World");
        myHT.Add("Third", "!");

        // Displays the properties and values of the Hashtable.
        Console.WriteLine("myHT");
        Console.WriteLine("  Count:    {0}", myHT.Count);
        Console.WriteLine("  Keys and Values:");
        PrintKeysAndValues(myHT);
    }
    public static void PrintKeysAndValues(Hashtable myList)
    {
        IDictionaryEnumerator myEnumerator = myList.GetEnumerator();
        Console.WriteLine("\t-KEY-\t-VALUE-");
        while (myEnumerator.MoveNext())
            Console.WriteLine("\t{0}:\t{1}", myEnumerator.Key, myEnumerator.Value);
        Console.WriteLine();
    }
}
```

SortedList

```
using System;
using System.Collections;
public class SamplesSortedList
{
    public static void Main()
    {
        // Creates and initializes a new SortedList.
        SortedList mySL = new SortedList();
        mySL.Add("First", "Hello");
        mySL.Add("Second", "World");
        mySL.Add("Third", "!");
        // Displays the properties and values of the SortedList.
        Console.WriteLine("mySL");
        Console.WriteLine("  Count:    {0}", mySL.Count);
        Console.WriteLine("  Capacity: {0}", mySL.Capacity);
        Console.WriteLine("  Keys and Values:");
        PrintKeysAndValues(mySL);
    }
    public static void PrintKeysAndValues(SortedList myList)
    {
        Console.WriteLine("\t-KEY-\t-VALUE-");
        for (int i = 0; i < myList.Count; i++)
        {
            Console.WriteLine("\t{0}:\t{1}", myList.GetKey(i), myList.GetByIndex(i));
        }
        Console.WriteLine();
    }
}
```

Stack (1)

```
//using System.Collection.Generics;
//Empleados
System.Collections.Stack employees = new Stack();
//Use el método Push del Stack
//El parámetro del un tipo de objeto
//Usa conversion implicita
employees.Push(new Employees());
//Use el método Pop del Stack
//Retorna el tipo de objeto
//Necesita conversion explicita
Employees employee = (Employees)employees.Pop();
//otro ejemplo con Integers
System.Collections.Stack sizes = new Stack();
//Box
sizes.Push(42);
//Unbox
int size1 = (int)sizes.Pop();
//Incorrecto, pero compila bien
sizes.Push(77);
sizes.Push(new Employees());
//Compila bien pero genera un
//InvalidCastException
int size2 = (int)sizes.Pop();
```

Stack (2)

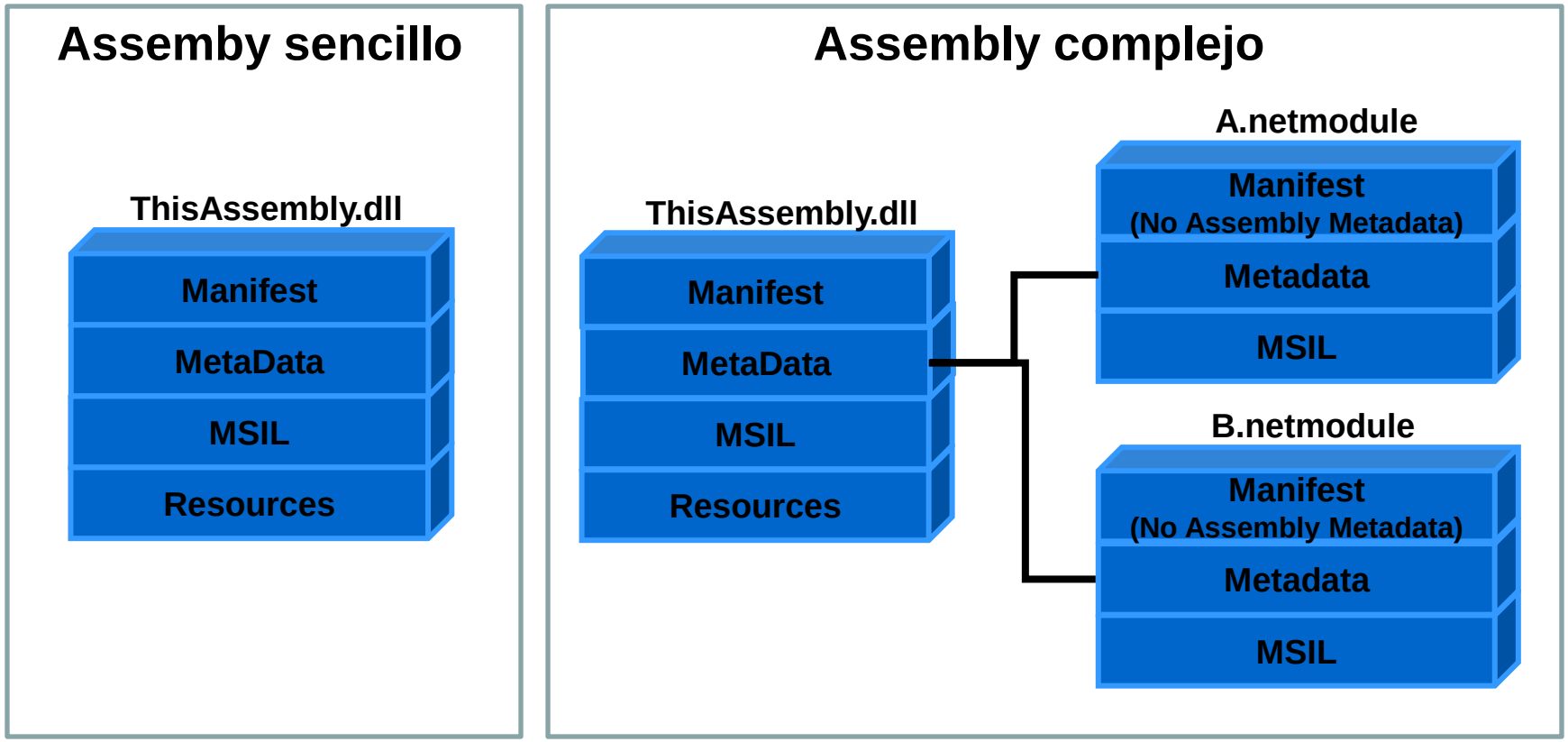
```
Stack<Employees> employees = new Stack<Employees>();
//Uso el método Push del Stack
//El parámetro el del tipo Employee
//No requiere conversion
employees.Push(new Employees());
//Uso el método Pop del Stack
//El retorno es del tipo Employee
//No requiere conversion
Employees employee = employees.Pop();

Stack<int> sizes = new Stack<int>();
//No boxing
sizes.Push(42);
//No unboxing
int size1 = sizes.Pop();
sizes.Push(77);
//Error!
//Causa una error en la compilación
sizes.Push(new Employees());
//Ahora se ejecuta correctamente
int size2 = sizes.Pop();
```

BIOS

Assembly

Definición

- Un ***Assembly*** es la unidad mas pequeña de distribución de código, instalación y versionado.



Caracteristicas

- ***Unidad de distribución***:
  - Uno o mas archivos independientemente del empaquetado (packaging)
  - Auto descriptivo usando metadata: Reflection
- ***Versionado*** : capturada por el compilador
- ***Frontera de Seguridad:***
  - Contienen permisos de ejecución
  - Los métodos pueden demandar pruebas de permisos concedido para todo el proceso de ejecución.

Metadata

- Descripción del assembly
  - Identifica nombre, versión, cultura, llaves publicas
  - Que tipos son exportados
  - A que otros ***assembly*** se hace referencia
  - Permisos que se necesitan para la ejecución
- Descripción de tipos
  - Nombre, visibilidad, clases base, interfaces que implementa
  - Miembros (métodos, campos, propiedades, eventos, tipos anidados)
- Sentencias declarativas
  - Atributos definido por el usuario
  - Atributos definidos por el compilador
  - Atributos definidos por el Framework

MSIL

- MicroSoft Intermediate Language
- El MSIL es independiente del lenguaje en el que se desarrolla.
- Cuando el código administrado (C#, VB.NET, etc.) es compilado, se genera un ***assembly*** (archivo .Dll o .Exe) conteniendo:
  - Código MSIL
  - Metadata
- El MSIL es compilado a código nativo por el JIT antes de que sea ejecutado.

Global Assembly Cache

Definición

- Cache a nivel máquina.
- Almacena ***assemblies*** que deben ser compartidos por diferentes aplicaciones.
- Instalar en GAC solo los ***assemblies*** que deben ser compartidos y mantener privados a los que no.
- Para instalar en GAC un ***assembly***:
  - Utilizar un instalador
  - Utilizar la herramienta gacutil.exe
  - Utilizar el Windows Explorer para arrastrar los ***assemblies*** al cache

Hilos de Ejecución (Threading)

Descripción

- Tradicionalmente, los desarrolladores que trabajan creaban aplicaciones sincrónicas que ejecutan tareas en forma secuencial.
- Los programas de subprocesos(threads) múltiples son posibles debido a las tareas múltiples.
- Cada Subproceso tiene un costo en recursos.
- Demasiados Subprocesos pueden reducir el rendimiento.
- Espacio de nombre ***System.Threading.***
- El método proporcionado como argumento para el método ***Thread*** no puede tener un parámetro o valor de retorno

Ventajas

- Establece configuraciones de prioridad para optimizar el rendimiento.
- Se usa para controlar la orden de ejecución de códigos.
- Está mejor adecuado para:
  - tareas que consumen mucho tiempo o requieren un arduo trabajo del procesador y bloquean la interfaz.
  - tareas que esperan un recursos externo como un archivo remoto o la conexión a Internet.

Temporizador

- La clase ***Threading.Timer*** es útil para ejecutar periódicamente una tarea en un subproceso por separado o establecer una ejecución demorada. Se utiliza junto con el delegado ***TimerCallback***
- Dicha clase se puede configurar para operación de una sola vez o continúa.
- La clase ***Callback*** se ejecuta utilizando el subproceso del grupo de subprocesos CLR
- Es útil cuando la clase ***System.Windows.Forms.Timer*** no está disponible.

Ejemplo (1)

```
using System.Threading;

namespace ConsoleApplication2
{
    class Program
    {
        public static void mimetodo()
        {
            //implementación
        }
        static void Main(string[] args)
        {
            Thread hilo = new Thread(new ThreadStart(mimetodo));
            hilo.Start();
        }
    }
}
```

Ejemplo (2)

```
static void Main(string[] args)
{
    Thread hilo = new Thread(new ThreadStart(mimetodo));
    hilo.Start();
    hilo.Join(); //Esperar a que finalice el proceso
    Console.WriteLine("El subproceso ha finalizado");
    Console.ReadLine();
}
```

Ejemplo (3)

```
class Program
{
    public static void mimetodo2(object param)
    {
        Console.WriteLine("método2");
    }
    public static void mimetodo(object param)
    {
        Console.WriteLine("método1");
    }
    static void Main(string[] args)
    {
        ThreadPool.QueueUserWorkItem(new WaitCallback(mimetodo));
        ThreadPool.QueueUserWorkItem(new WaitCallback(mimetodo2));
        Thread.Sleep(1000);
        Console.WriteLine("El Main thread ha finalizado");
        Console.ReadLine();
    }
}
```