A.) Teoria .NET

1. ¿Qué es Microsoft Visual Studio .Net?

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para Windows y macOS. Es compatible con múltiples lenguajes de programación, tales como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby y PHP, al igual que entornos de desarrollo web, como ASP.NET MVC, Django, etc., a lo cual hay que sumarle las nuevas capacidades en línea bajo Windows Azure en forma del editor Monaco.

Visual Studio permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno compatible con la plataforma .NET (a partir de la versión .NET 2002). Así, se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos y videoconsolas, entre otros.

2. ¿Qué es el Framework .Net?

Cuando hablamos de .NET Framework, estamos hablando de este conjunto de estructuras y tecnologías que proporciona Microsoft para una programación más sencilla orientada a las redes e internet, con independencia de la plataforma hardware utilizada. Microsoft .Net Framework es un componente software que da soluciones de código que son utilizadas por los programas (a través de las librerías dll), y que gestiona programas escritos para este Framework.

.Net Framework proporciona un entorno de desarrollo que utiliza un software sencillo, aumentando la seguridad de los programas y reduciendo las vulnerabilidades.

Los principales componentes del entorno de trabajo .Net recogen la biblioteca de clases base, los lenguajes de programación y el entorno común para ejecución de lenguajes (CLR).

3. ¿Qué es el CLR?

El Common Language Runtime o CLR ("entorno en tiempo de ejecución de lenguaje común") es un entorno de ejecución para los códigos de los programas que corren sobre la plataforma Microsoft .NET. El CLR es el encargado de compilar una forma de código intermedio llamada Common Intermediate Language (CIL, anteriormente conocido como MSIL, por Microsoft Intermediate Language), al código de máquina nativo, mediante un compilador en tiempo de ejecución. No debe confundirse el CLR con una máquina virtual, ya que una vez que el código está compilado, corre nativamente sin intervención de una capa de abstracción sobre el hardware subyacente.

4. ¿Qué es la BCL?

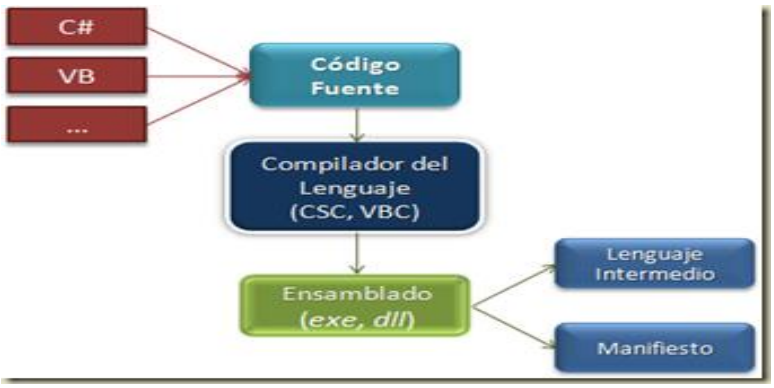
Su traducción al español sería la librería de clases base (Base class library). La BCL está formada por bibliotecas o APIs especializadas que pueden ser utilizadas por todos los lenguajes de programación de la plataforma .NET.

Cada una de estas bibliotecas puede contener a su vez numerosas clases que aglutinan varios métodos y funciones con características concretas.

5. Indique y explique el orden que se sigue en el proceso de compilación y ejecución en .net

Todo ensamblado está implementado a partir de un lenguaje de programación soportado por el .NET Framework: C#, Visual Basic, etc., cada uno de estos lenguajes, sólo proporciona su sintaxis y compilador para convertir el código fuente a un archivo compilado en lenguaje intermedio

(Intermediate Language), el cual puede ser un archivo EXE ó DLL (ensamblado). En el siguiente gráfico puede observar el proceso de compilación.



Una vez generado el ensamblado, será desplegado hacia la PC del usuario o servidor, donde el .NET Framework se encuentra instalado como prerrequisito.

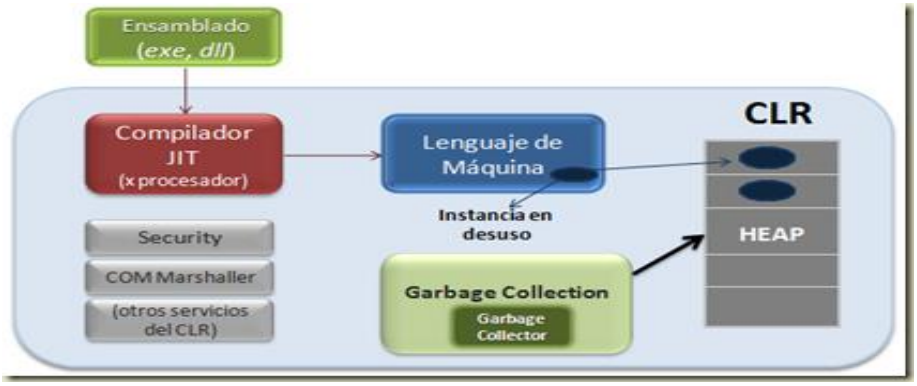
Luego de desplegar el ensamblado, su ejecución queda bajo responsabilidad del .NET Framework, el cual, proporciona una serie de servicios que habilitan un ambiente seguro para la ejecución (managed code) dentro del CLR (Common Language Runtime), entre los que podemos nombrar:

El Motor de ejecución JIT (Just In Time Activation), este servicio, se encarga de tomar la porción de código que necesita ser ejecutada (On Demand), y lo convierte a lenguaje de máquina, esto es necesario, debido a que el procesador donde ejecuta el ensamblado, puede ser diferente al que se usó para su construcción.

Garbage Collection: este proceso se encarga de ir monitoreando los recursos que ya no son empleados por la aplicación, para ser depurados a partir de Heap del sistema (repositorio de las instancias de objetos en desuso), o también cuando la aplicación requiere recursos, aquí es importante recalcar que el objeto responsable de liberar los recursos es el Garbage Collector (GC), el cual NO DEBE ser ejecutado desde la aplicación por motivos de performance.

Seguridad: esta característica, permite controlar el acceso a los recursos del ensamblado, ya sea a partir de la identidad de otro ensamblado (Code Access Security – CAS), o la entidad de un usuario (Role Base Security – RBS), para lo cual, un ensamblado debe tener asociado un Nombre Seguro (Strong Name) de esto hablaremos más adelante.

En el siguiente gráfico podemos observar el trabajo del CLR durante la ejecución de un ensamblado.



6. ¿Cuál es la signatura de Main?

La signatura o firma de un método o una función define su entrada y su salida. Incluye por lo menos el nombre de la función o método y el número de sus parámetros. En algunos lenguajes de programación, puede incluir el tipo que devuelve la función o el tipo de sus parámetros.

El método Main es el punto de entrada de una aplicación de C# (las bibliotecas y los servicios no requieren un método Main como punto de entrada). Cuando se inicia la aplicación, el método Main es el primero que se invoca.

Solo puede haber un punto de entrada en un programa de C#. Si hay más de una clase que tiene un método Main, debe compilar el programa con la opción del compilador -main para especificar qué método Main desea utilizar como punto de entrada.

```
class TestClass
{
    static void Main(string[] args)
    {
        // Display the number of command line arguments.
        Console.WriteLine(args.Length);
    }
}
```

En los ejemplos siguientes se usa el modificador de acceso público. Es el procedimiento habitual, pero no es necesario.

Al agregar los tipos de valor devuelto async, Task y Task<int>, se simplifica el código de programa cuando las aplicaciones de consola tienen que realizar tareas de inicio y await de operaciones asincrónicas en Main.

```
public static void Main() { }
public static int Main() { }
public static void Main(string[] args) { }
public static int Main(string[] args) { }
public static async Task Main() { }
public static async Task<int> Main() { }
public static async Task Main(string[] args) { }
public static async Task<int> Main(string[] args) { }
```

7. ¿Qué es un espacio de nombre (namespace)?

En programación, un espacio de nombres (técnica y correctamente definido como namespace), en su acepción más simple, es un conjunto de nombres en el cual todos los nombres son únicos.

8. ¿Cómo se incluye una librería en C#?

Sintaxis para declarar Librerías en C#, por ejemplo: “Using System.Data”

Using: Es el comando para llamar la Librería.

System: En este caso es el espacio de nombre que posee la librería que necesito.

Data: Es la librería.

Las librearías son archivos (no siempre externos) que nos permiten llevar a cabo diferentes tareas sin necesidad de preocuparnos por cómo se hacen sino simplemente entender cómo usarlas. Las librerías en C# permiten hacer nuestros programas más modulares y reutilizables,

facilitando además crear programas con funcionalidades bastante complejas en unas pocas líneas de código. También podemos crear nuestras propias librerías con el fin de reutilizar código útil.

9. ¿Qué indica la palabra clave params?

Mediante el uso de la palabra clave params, puede especificar un parámetro de método que toma un número variable de argumentos. El tipo de parámetro debe ser una matriz unidimensional.

No se permiten parámetros adicionales después de la palabra clave params en una declaración de método, y solo se permite una palabra clave params en una declaración de método.

Cuando se llama a un método con un parámetro params, se puede pasar:

- Una lista separada por comas de argumentos del tipo de los elementos de la matriz.
- Una matriz de argumentos del tipo especificado.
- Sin argumentos. Si no envía ningún argumento, la longitud de la lista params es cero.

D) OBJETOS:

1. ¿Qué es una clase?

2. ¿Qué es un objeto?

3. Porque se caracterizan los objetos?

4. ¿Cómo se llama la táctica de obtener la forma mínima y esencial de un objeto?

1. Para el programador orientado a objetos, una **Clase** es una construcción sintáctica con nombre que describe un comportamiento y atributos comunes. Una estructura de datos que incluye datos y funciones.

La palabra clase proviene de clasificación. Formar clases es el acto de clasificar, algo que hacen todos los seres humanos (y no sólo los programadores). Por ejemplo, todos los coches comparten un mismo comportamiento (se pueden dirigir, detener,…) y atributos comunes (tienen cuatro ruedas, un motor,…). Usamos la palabra coche para referirnos a todos estos comportamientos y propiedades comunes.

2. Un **Objeto** es una instancia de una clase. La palabra coche significa distintas cosas según el contexto. A veces usamos la palabra coche para referirnos al concepto general de coche: hablamos de coche como una clase, la del conjunto de todos los coches, sin pensar en ningún tipo de coche concreto. En otras ocasiones empleamos la palabra coche para hablar de un coche en particular. Los programadores usan el término objeto o instancia para referirse a un coche concreto. Es importante entender esta diferencia.

3. Los objetos se caracterizan por:

Identidad: Los objetos se distinguen unos de otros.

Comportamiento: Los objetos pueden realizar tareas.

Estado: Los objetos contienen información.

4. La táctica para obtener la forma mínima y esencial de un objeto se llama “instanciar”.