1. ¿Que es .Net framework y cómo funciona?

Es una máquina virtual que ejecuta código manejado, código que es ejecutado por el CLR (Common Language Runtime) y que ha sido compilado a partir de C# o VB .Net.

Funciona de la siguiente manera:

Haces un programa en C# o VB .Net y lo compilas, este se traduce a CIL (Common Intermediate Language).Posteriomente, el programa en ensamblado a bytecode para generar un archivo ensamblado CLI (Common Language Infrastructure) que puede ser .exe o .dll. Corres el programa (o utilizas la DLL) y este es ejecutado por el .Net framework CLR (Common Language Runtime), no por el sistema operativo directamente, por esta razón se le llama "Código Manejado" (Managed Code). El .Net Framework CLR, mediante el compilador JIT (Just In Time Compiler), se encarga de compilar este código manejado en lenguaje intermedio a lenguage máquina nativo ensamblador, para que pueda ser ejecutado por el CPU. El CIL (Common Intermediate Language) es el lenguage que entiende .Net Framework. C# y VB .Net son lenguages que comprendemos los humanos, C# y VB. Net son traducidos a CIL.

2. ¿Que es el Heap y Que es el Stack?

Ambas son ubicaciones de memoria, el Heap es global, el Stack es local. El Heap es a nivel de aplicación, el Stack a nivel de hilo. El Stack es una pila, primeras entradas primeras salidas, el Heap no tiene una estructura de datos definida. Su tamaño se define al momento de su creación, el Heap al iniciar la aplicación, el Stack al crear un hilo. Ambos pueden crecer dinámicamente. El Stack es más rápido, es una pila, esta en "cache", no tiene que sincronizarse con los demás hilos como el Heap. El Stack almacena valores, el Heap, objetos.

3. ¿Que es el Garbage Collector?

Es una proceso automático de liberación de memoria. Cuando la memoria se esta agotando, para los hilos que esten corriendo, recorre el Heap y elimina los objetos que ya no estan siendo utilizados, libera memoria, reaorganiza todos los hilos que quedan y ajusta los pointers a estos objetos, tanto en el Heap como en el Stack.

4. ¿Que es un delegate?

Es una definición de un método, encapsula determinados argumentos y tipo de retorno. Permite pasar un método como argumento de una función, siempre y cuando respete la definición.

5. ¿Que es LINQ?

Es una estandarización para consultar datos y convertirlos en objetos, independientemente de la fuente. Es un administrador de consultas para bases de datos, xml y colecciones enumerables usando un solo lenguaje.

6. ¿Cómo funciona LINQ?

Internamente construye la consulta correcta (en el caso de las bases de datos) o genera las operaciones correspondientes sobre las colecciones o parsea el xml y devuelve los datos correspondientes. Encapsula todos estos comportamientos y proporciona una implementación única, de esta manera, podemos utilzar las mismas consultas, el mismo lenguaje, independientemente de la fuente de datos subyacente.

7. ¿Que es la ejecución diferida y la ejecución inmediate en LINQ?

La ejecución diferida es encapsular la definición de la consulta, pero no ejecutarla hasta que en tiempo de ejecución los datos sean utilizados. La ejecución inmediata mánda a llamar la consulta al mismo momento de su definición. Por defecto, las ejecuciones son diferidas, pero podemos hacerlas inmediatas al llamar ".ToList()" por ejemplo, de esta manera, se ejecutará y nos devolverá una lista de objetos al momento en que la definamos.

8. ¿Que es un objeto y una clase?

Un objeto es una instancia de una clase y una clase es una plantilla para crear objetos, es la definición de un objeto, la descripción de sus características y operaciones, de sus propiedades y sus métodos. Un objeto tiene identidad por que sus características tienen valores.

9. ¿Que es herencia, poliformismo y encapsulación?

Herencia es la capacidad de reutilizar definiciones de una clase en otra, es poder basar una clase en otra. Poliformismo es poder declarar el mismo método dentro de una clase con diferentes argumentos o diferente tipo de retorno. Encapsulación es poder exponer solamente los métodos, propiedad y argumentos necesarios para utilizar las operaciones de una clase, mientras que su implementación detallada permanece privada, escondida a otros objetos.

10. ¿Cuál es la diferencia entre una clase abstracta y una interface?

La clase abstracta puede contener constructores, métodos y campos, públicos y privados. La interface solo métodos y propiedades públicas. Solo puedes hererar de una clase abstracta, pero implementar de muchas interfaces. Una interface define comportamiento, algo que la clase que la implemente podrá hacer. Una clase abstracta define lo que la clase es, lo que representa. Ninguna de las dos puede ser instanciada. Una clase abstracta es útil al crear componentes, para hacer una implementación inicial parcial y una definición concreta, dejándo libertad para implementar otros métodos.

11. Diferencia entre modificadores public y estáticos.

Un método, campo o propiedad estático puede ser invocado sin tener que instanciar la clase.Un método público debe ser invocado desde una instancia de una clase.

12. ¿Que es una clase sellada (sealed class)?

Es una clase que no puede ser heredada. Se utiliza cuando, por diseño, una clase esta super especializada y no debe ser modificada por sobreescritura (overriding).

13. ¿Que es un "jagged array"?

Es un arreglo de arreglos.

14. ¿Que es serialización?

Convertir un objeto a una corriente de datos. Debe implementar ISerialize.

15. ¿Cuál es la diferencia entre constantes y variables de solo lectura?

Constantes son declaradas e inicializadas en la compilación. Su valor no puede cambiar. Las variables de solo lectura son asignadas en tiempo de ejecución.

16. Explica Mutex

Mutuamente exclusivo es un manejador de recursos compartidos por hilos, se asegura que solo un hilo a la vez haga uso de un recurso (un objeto) a la vez. Es como un moderador que controla el microfono y cede la palabra a una persona a la vez, así, Mutex concede acceso a los recursos un hilo a la vez, poniendo “en espera” a los recursos que quieren acceder a los recursos, hasta que estos esten desocupados.

17. ¿Que es inmutabilidad, para que sirve y como se codifica?

Capacidad de los objetos de no cambiar su estado después de ser creados, sirve para mejorar la mantenibilidad del código ya que un objeto mutable puede encapsular sus cambios y no quedar explícito en el código, haciendo difícil seguir el flujo, especialmente en las aplicaciones multi hilos. Para crear objetos inmutables, pasas los argumentos para su creación en el constructor y posteriormente haces sus propiedades read-only.

18. ¿Cual es la diferencia entre Override y Overload en un método?

Override es sobreescribir el método, mismo firma (parámetros y tipo de retorno) pero diferente funcionalidad. El método debe ser declarado “virtual” para poder sobreescribirlo. Overload es codificar varias versiones de un mismo método, siempre que tenga diferente firma (parámetros y/o valor de retorno). No es necesario que el método sea “virtual” para aplicarle overloading.

19. ¿Cuál es la diferencia entre struct y class?

Una clase es una definición de un objeto y puede ser heredada. Una estructura define un tipo de datos, y no puede ser heredada.

20. ¿Cuál es la diferencia entre ODBC and ADO?

Open DataBase Connectiviy es un estandar para manejar operaciones de bases de datos en aplicaciones, un estandar, utilizando los mismos métodos para Oracla que para Mysql, por ejemplo, con la particularidad que se declaran las conexiones a nivel de usuario o sistema operativo. ADO es una conjunto de librerías .Net para el manejo de datos, incluyendo conexiones ODBC. Para ADO, ODBC es un driver.

21. ¿Cuál es la diferencia entre encriptar un password y aplicarle un hashing?

El hashing (MD5 o SHA1, por ejemplo) no puede desencriptarse, para validar el password debemos contar con el password en texto plano, aplicarle un hash y compararlo contra el almacenado. Un password encriptado, si contamos con las llaves y conocemos el algoritmos de encriptación (como Triple-DES por ejemplo) podemos obtener el password en texto plano a partir del password encriptado.

22. ¿Que es Reflection y para que sirve?

Es la capacidad de leer propiedades y métodos de las clases de un ensamble, para poder instanciarlas y e invocarlas. Es especialmente útil cuando no contamos con el código fuente original de estas clases, solo su ensamblado.

23. ¿Que es una patrón de diseño y para que sirve? Da algunos ejemplos

Es una plantilla reusable para resolver un problema común a nivel de diseño. No es código, sino mejores prácticas para codificar una solución. Algunos ejemplos son Singleton, Abstract Factory, Observer o Pub/Sub, Modelo Vista Controlador, Modelo Vista Presentador y Modelo-Vista Vista-Modelo.

24. ¿Para que utilizamos la declaración “using”?

Using se utiliza para asegurarse de liberar los recursos del objeto utilizado, ya que siempre llama “Dispose” cuando el termina su bloque de código.

25. ¿Que es una variable de tipo implicita y cual es su alcance?

Es una variable para cual no tienes que declarar tipo, ya que el compilador determina automáticamente su tipo. Su alcance es local, dentro de un método y solo permite inferir el tipo la primera vez que se le asigna un valor, a la segunda, si el tipo es diferente, lanzará un error.

26. ¿Que es método anónimo y como se diferencia de una expresión lamba?

Un método anónimo es método que no necesita nombre ni ser declarado antes de ser usado, implementado a través de un delegado. Una expresión lambda es una definición de un método anónimo en una sola línea, reemplazando al delegado para esta función, en una forma más elegante.

27. ¿Qué es Native Image Generator?

Es una herramienta que compila los ensambles de .Net a codigo máquina para un procesador específico, de esta manera mejora su desempeño, pues el JIT ya no interviene.

28. ¿Es el JIT un intérprete?

No, el JIT no es un intérprete, es un compilador en tiempo de ejecución que mejora el desempeño compilando método por método solo una vez, ya que si el método se llama de nueva cuenta, se utiliza el código nativo ya compilado, mientras que un intérprete ejecuta el mismo cada bloque de código.