**CLASE 01: Cuestionario Teórico: C#**

1. ¿Qué es el CLR?

Cuando compilamos un programa desarrollado en plataforma .NET, se compila a un lenguaje denominado MSIL (Microsoft Intermediate Lenguage), el cual es traducido a lenguaje maquina mediante el CLR (Common lenguage Runtime).

El CLR es una máquina virtual que compila / traduce el código MSIL en tiempo de ejecución. “Su funcionamiento es muy parecido al hipervisor de una máquina virtual, lo cual permite ejecutar aplicaciones .NET multiplataforma.

A su vez, el CLR también se encarga de reservar espacio de memoria y liberar recursos del sistema una vez finalizada la ejecución de la aplicación gracias al garbage collector.

1. Describa el proceso de compilación de C#.

El código fuente, gracias al compilador de lenguaje, es traducido a un código intermedio (IL) ensamblado con metadatos, los cuales son tablas que contienen información / descripción completa del código.

Gracias a la metadata, el compilador JITTER (el cual se encuentra dentro del CLR), obtiene toda la información que este necesita para compilar el código a lenguaje máquina.

1. ¿Qué es el CTS?

El Common Type System (CTS) es el conjunto de reglas que se deben seguir en las definiciones de tipos de datos para que el CLR las acepte. En otras palabras, aunque cada lenguaje tiene su propia sintaxis para definir tipos de datos, en el MSIL resultante de la compilación de su código fuente se tienen que cumplir las reglas del CTS.

1. ¿Qué es un tipo por referencia (Reference Type)? ¿En qué se diferencia de un tipo por valor (Value Type)?

Una variable de un tipo de valor contiene un valor de un tipo determinado. (Entero, Flotante, etc). Estos tipos de variables se eliminan de la memoria stack cuando el método en donde se la está utilizando termina su ejecución.

Cuando se pasa una variable por valor, se pasa una copia de la misma y ésta también se almacena en la memoria stack.

Una variable por referencia en cambio contiene la dirección de memoria de donde se encuentran los datos de manera que se usa esa dirección para acceder de forma indirecta a los mismos. Se asigna en la memoria Heap y tiene la ventaja de poder ser accesible desde otras clases.

1. Explique las diferencias entre variables escalares y no escalares.

Un escalar es un determinado valor (que no varía todo a lo largo del programa informático), una variable, o un campo, que solamente puede tener un valor en un cierto momento; en comparación o contraposición, están los conceptos de array (vector), lista y objeto, que pueden tener almacenado en su estructura más de un valor.

1. ¿Cuál es el punto de entrada (entry point) para los programas en C#?

El entry point es un solo método público estático MAIN que recibe un array de tipo string como parámetro opcional.

1. ¿Cuál es la diferencia entre una conversión implícita y una explícita?

En una conversión implícita no interviene el programador y no hay perdida de datos y en la explícita sí puede haber perdida de datos (por ejemplo pasar de float a int y perder la parte decimal) e interviene el programador, es decir, se tiene que castear.

¿Para qué sirven los modificadores out y ref? ¿En qué se diferencian?

Aunque ambas hacen que se pase una variable por referencia y no por valor, ref indica que el parámetro es de entrada/salida.

Out, es únicamente de salida y se puede utilizar en caso de necesitar varios valores de retorno (como en el método TryParse, donde obtenemos un valor booleano en el return tradicional y un valor parseado en el parámetro out).

Ésto nos lleva a una diferencia importante: los parámetros pasados como ref deben estar inicializados previamente (se va a modificar su valor), mientras que los parámetros pasados como out no es necesario que lo estén (se va a asignar un valor sin importarnos lo que tuviera previamente).

Explique cómo funciona el método TryParse. ¿Qué retorna? ¿Qué recibe? ¿Cuál es la diferencia con el método Parse?

Ambos métodos se usan para convertir un string a un tipo deseado por el programador.

**TipoDeDato.Parse():** Recibe como parámetro el string a convertir, y retorna una variable del tipo deseado. En el caso de que no haya podido hacerlo, puede devolver 3 excepciones según que parámetro se le haya pasado. (Parametro Nulo, Parametro con Error de formato y Parametro fuera de los limites del tipo deseado como retorno)

**TipoDeDato.TryParse():** Recibe como parámetro un string a convertir, una dirección de memoria del tipo deseado y retorna un valor booleano.

En el caso de que la conversión haya sido exitosa, retorna TRUE y guarda el valor convertido en la dirección de memoria mencionada, caso contrario, retorna FALSE.

1. ***¿Qué propone el Paradigma orientado a Objetos? ¿Qué es un Paradigma?****Un paradigma consiste en un método para llevar a cabo cómputos y la forma en la que deben estructurarse y organizarse las tareas que debe realizar un programa. El paradigma orientado a objetos propone resolver problemas de la realidad a través de identificar objetos y relaciones de colaboración entre ellos..*
2. ***¿Nombre los pilares del POO?****Los pilares son: Abstracción: Decide que es lo importante y que no lo es, se enfoca en lo que es importante y lo que no lo es simplemente lo ignora.  
   Encapsulamiento: Es la que denota la capacidad del objeto de responder a peticiones a través de métodos y propiedades sin necesidad de exponer los medios utilizados.  
   Herencia: Los objetos pueden heredar métodos y datos de otros. Esto hace más fácil definir nuevos objetos a partir de otros que ya teníamos y facilita la reescritura de programas.  
   Polimorfismo: Un mismo nombre de un método puede hacer referencia a comportamientos distintos.*
3. ***¿Qué es una clase?****Una clase es una clasificación en base a comportamientos y atributos comunes, a partir de esta se crea un vocabulario. Es una abstracción de un objeto.*
4. ***¿Qué es un objeto? ¿En que se relacionan con las clases?****Los objetos son clases instanciadas, se crean en tiempo de ejecución y poseen comportamientos (métodos) y estados (atributos). Es una entidad concreta basada en una clase.*
5. ***¿Qué significa instanciar un objeto?****Instanciar un objeto es reservar espacio de memoria de un tipo de clase para ese objeto y se hace mediante la palabra reservada New. La variable contiene solo una referencia a la ubicación de dicho objeto.*
6. ***Explique que es la abstracción en el contexto de la programación orientada a objetos y cual es su relación con las clases.****La abstracción son aquellas características especificas de un objeto, las que lo distinguen de los demás y que logran definir límites conceptuales respecto a quien esta haciendo dicha abstracción, es básicamente el principal concepto para la creación de una clase.*
7. ***¿Qué son los atributos o estado de un objeto?****Los atributos son las variables y determinan las características de un objeto, determinan su apariencia, estado u otras cualidades.*
8. ***¿Qué son los métodos de un objeto?****Los métodos de instancia son las funciones de los objetos que determinan los comportamientos o que pueden hacer, generalmente una serie de sentencias para llevar a cabo una acción.*
9. ***¿Qué es un método estático, en qué se diferencia de los métodos de instancia?****Un método estático es un “Método de clase”, puede usarse sin necesidad de instanciar el objeto. No puede acceder a los atributos de la clase (Salvo que sean estáticos), no puede utilizar el operador “this”, y solo puede llamar a otros métodos siempre y cuando sean estáticos.*
10. ***¿Qué es una clase estática, en que se diferencia de una no-estática?****Una clase estática es aquella que se puede usar sin necesidad de realizar una instancia de la misma, cuando se declara una clase estática sus atributos y métodos tienen que ser estáticos. Además se llama 1 sola vez a su constructor estático.*
11. ***¿Puedo tener miembros estáticos en clases no estáticas, y viceversa?****Si se puede tener miembros estáticos en clases no estáticas, pero las clases estáticas solo tienen miembros estáticos.*
12. ***¿Necesito instanciar un objeto de la clase para llamar a sus métodos estáticos?****No, no es necesario.*
13. ***¿Cuántos valores distintos pueden existir para un campo estático y cuántos para uno no estático si se crearon 5 instancias de esa clase?****Si el campo es estático el valor va a ser único para todas las instancias ya que ese campo es el mismo para todas las instancias, en cambio si el campo no es estático puede tener valores distintos para cada instancia.*
14. ***¿Se puede usar el operador “this” dentro de un método estático? ¿Se puede acceder a miembros no-estáticos desde un método estático de la misma clase, y viceversa?****No, no se puede utilizar el operador this, y solo puede acceder a miembros estáticos. Desde un método de instancia si se puede acceder a miembros estáticos.*
15. ***¿Se puede declarar variables estáticas dentro de un método (locales)?****No ya que las variables estáticas están hechas para tener un tiempo de vida largo y durar hasta la finalización de ejecución del programa.*
16. ***De dos ejemplos de métodos estáticos que pertenezcan a las clases .NET Framework****System.Math.Abs y System.Console*
17. ***Lo mismo que antes pero no estáticos.****StringBuilder y Random.*
18. ***¿Qué es un namespace y cuál es su función principal?****Un namespace es una agrupación lógica de clases y otros elementos, toda clase está dentro de un Namespace. Proporcionan un marco de trabajo jerárquico sobre el cual se construye y organiza todo lo lógico. Su función principal es la organización del código para reducir conflictos entre nombres.*
19. ***¿Puedo tener distintos namespaces dentro de un mismo proyecto o ensamblado de .NET?****Si se pueden tener distintos namespaces.*
20. ***¿Para que se usa la directiva using?****Permite la especificación de una llamada a un método sin el uso obligatorio de un nombre completamente cualificado.*
21. ***¿La directiva alias?****Permite utilizar un nombre distintos para un Namespace. Se utiliza para abreviar nombres largos.*
22. ***¿Puedo declarar dos clases independientes/distintas con el ismo identificador dentro del mismo namespace? ¿Y en namespace distintos?****Dentro del mismo namespace no, ya que habría conflicto de nombres, pero en distintos namespace si.*
23. ***¿Los objetos se crean en tiempo de?****Se crean en tiempo de ejecución (runtime) y es el intervalo de tiempo en el que un programa de computadora se ejecuta en un sistema operativo.*
24. ***¿Cuántos tipo Alumno puedo tener en mi sistema? ¿Cuántas clases Alumno tengo en mi sistema dentro de un mismo namespace?****Se puede tener tantos objetos Alumno como se desee, y dentro de un mismo namespace se puede tener solamente una clase Alumno*
25. ***Explique el ciclo de vida de un objeto.****Se compone de 3 partes:  
    - Primero se crea el objeto utilizando el operador New(el cual asigna memoria) y se usa un constructor para inicializar el objeto en esa memoria.  
    - Segundo se utiliza, se llaman a métodos y atributos.  
    - Tercero se destruye, la referencia en memoria se pierde (porque termino el programa, se cambió o eliminó la variable, etc). El Garbage Collector liberará memoria cuando lo crea necesario.*
26. ***¿En que segmento de memoria se almacenan los datos tipo valor (value type) y en cuál los tipos referencia (reference type)?****El CLR administra dos segmentos de memoria, Stack y Heap. Los tipos valor (short, char, double, string, int, etc.) se almacenan en el Stack y los de referencia en el Heap.  
    El GC administra el segmento de memoria Heap, ya que lo almacenado en el Stack se elimina cuando termina la función que esté utilizando el almacenado en el mismo.*
27. ***Compare y describa destrucción determinista y no determinista.****Determinista: Una variable local se crea en el momento de declararla y se destruye al final del ámbito (scope) en el que está declarada. Tiene un punto inicial y final de su vida.  
    No determinista: Un objeto aparece cuando se crea pero no se destruye al final del scope en el que se crea, su creación es determinista pero su destrucción no, ya que puede existir mucho más allá de una llamada al método.*
28. ***¿Cuál es la diferencia entre declarar, instanciar e inicializar un objeto?****Declarar es hacer la implementación lógica de dicho objeto, es decir codearlo. Instanciarlo es hacerle un New e inicializarlo es llamar a algún Constructor.*
29. ***¿Qué es una constante? ¿Se puede declarar constantes estáticas (static const)?****Una constante es una variable que tendrá el mismo valor siempre, y una variable estática tendrá el mismo valor para cualquiera que este llamándola, pero se podrá modificar el valor y se modificara para todas. Esto es por lo que no se puede crear una variable static constant.*
30. ***¿Cuándo se asigna el valor a las constantes? ¿Dos objetos del mismo tipo pueden tener distintos valores en una constante?****El valor es asignado en tiempo de compilación para permanecer inalterable durante el tiempo de ejecución, es por esto que no se puede obtener en variables objeto, distintos valores de una constante.*
31. ***¿Qué es un constructor?****Son métodos “especiales” ya que no retornan nada ni si quiera Void, que se utilizan para inicializar objetos y se usan junto con la palabra New.*
32. ***¿Qué es el constructor por defecto? ¿Qué sucede con el mismo cuando declaramos un constructor nuevo en la clase?****Es un constructor público, no recibe argumentos e inicializa los campos en 0, false o null. Si declaramos un nuevo constructor este reemplazara al constructor por defecto.*
33. ***¿Con qué valores se cargan los atributos cuando se llama al constructor por defecto?****Las variables númericas en 0, los string en cadena vacía, los objetos en null, y los booleanos en false.*
34. ***¿Qué es y para que sirve un constructor estático?****Un constructor estático inicializa los atributos estáticos, y es llamada una única vez cuando se instancia o utiliza el objeto, no llevan modificadores de acceso y tampoco pueden recibir parámetros.*
35. ***¿Cuántas veces se puede llamar a un constructor estático? ¿Quién lo puede llamar?****Es llamado una única vez por clase en una única ejecución del programa. El usuario no puede llamarlo de forma explícita y se lo llama implícitamente antes que se cree una instanciación o se use la clase para algo.*
36. ***¿Se ejecutara primero un constructor estático o uno de instancia?****Siempre se ejecuta primero el constructor estático.*
37. ***¿Qué significa sobrecargar un método o constructor?****Es cuando dos o más métodos en una clase comparten el mismo nombre pero con diferentes parámetros, número, tipo u orden, es decir se cambia la firma del método.*
38. ***¿Qué debe cambiar para que la sobrecarga sea válida?****Tiene que cambiar los parámetros, ya sea cantidad, tipo, u orden.*
39. ***¿La sobrecarga se resuelve en tiempo de ejecución o en tiempo de compilación? ¿Cómo se distingue a qué sobrecarga llamar?****Se resuelve en tiempo de compilación y se distingue por los parámetros que se le pasan al método o constructor.*
40. ***¿Se tiene en cuenta el nombre o identificador de los parámetros de entrada para una sobrecarga?****No, solo se tiene en cuenta el tipo, número u orden.*
41. ***¿Se tiene en cuenta el modificador de visibilidad para una sobrecarga?****No, se generará un error de sintaxis.*
42. ***¿Los métodos pueden tener el mismo nombre que otros elementos de una misma clase?*** *No, y en el caso que se desee habría que cambiar alguna letra por mayúscula o minúscula siguiendo alguna regla de estilo y no alterar por alterar.*
43. ***Dos razones por las cuales se sobrecargan métodos.****Si hay métodos similares que requieren parámetros diferentes.  
    Para añadir funcionalidad al código.*
44. ***¿Los métodos estáticos pueden ser sobrecargados?  
    Si.***
45. ***¿Agregar el modificador “static” es una sobrecarga válida?  
    No.***
46. ***¿Agregar un modificador “out” o “ref” en la firma del método sin cambiar nada más es una sobrecarga válida?  
    Si.***
47. ***¿Cambiar el tipo de retorno es una sobrecarga válida?  
    No.***
48. ***Si tenemos distintas sobrecargas de un método, ¿Cómo podemos reutilizar código?****Se pasan parámetros a los métodos más simples a los más complejos o viceversa, todo depende de los casos de productividad.*
49. ***¿Para qué se utiliza el operador “this()”?****Se utiliza para reutilizar código usando constructores ya creados, hace referencia a algún constructor.*
50. ***¿Se pueden sobrecargar los constructores estáticos?****No, ya que no recibe parámetros, es llamado automáticamente en tiempo de ejecución la primera vez que se usa o instancia esa clase.*
51. ***¿Se puede llamar a un constructor estático con el operador “this()”?****No, ya que este constructor se ejecuta automáticamente.*
52. ***¿Se puede llamar a constructores de otras clases con el operador “this()”?****No, solo de la misma clase.*
53. ***¿Se puede sobrecargar un constructor privado?****Si.*
54. ***¿Qué es un operador? ¿En qué se diferencia un operador unario y un operador binario?****Los operadores unarios realizan una acción con un solo operando. Los binarios realizan acciones con dos operando.  
    Ejemplo unario: ++, --, !, &, etc.  
    Ejemplo de operador binario: +, -, \*, /, etc.*
55. ***¿Qué varía en la sintaxis de la sobrecarga de operadores unarios y binarios?****La sobrecarga de operadores unarios recibe un solo argumento, generalmente del tipo de clase.  
    Los binarios deben recibir dos argumentos y uno tiene que ser del tipo de clase obligatoriamente.*
56. ***¿Se pueden sobrecargar los operadores de operación y asignación (+=, -=, etc) ¿Por qué?****No, ya que cuando se sobrecarga un operador binario se sobrecarga implícitamente el operador de asignación correspondiente.*
57. ***¿Cuál es la diferencia entre un operador de conversión implícito y uno explícito?****Los operadores de conversión permiten hacer compatibles tipos que antes no lo eran.**La diferencia es que el explicito se usa para Castear (T)x y el implicito se usa para inicializar T aux = x;*
58. ***Los operadores de casteo “(T)x” no se pueden sobrecagar. ¿Cuál es la alternativa?****Se deben definir nuevos operadores de conversión explícita.*
59. ***¿Cuál es la diferencia entre castear (casting), convertir(converting) y parsear (parsing)?****Parsear es convertir un string a un tipo de dato númerico.  
    Converting es cuando se fuerza la conversión y puede haber perdida de información.  
    Casting es cuando se avisa que se realizará una conversión y no hay perdida de información y si la hay es previamente sabido.*
60. ***¿Los formularios son objetos, de qué clase heredan todos los formularios?****Los formularios son objetos que exponen propiedades, métodos que determinan su comportamiento y eventos que definen la interacción con el usuario. Los formularios heredan de la clase Form.*
61. ***¿Qué es una partial class o clase parcial?****Una partial class permite separar el código de una clase en dos archivos fuentes diferentes.*
62. ***¿Puedo agregar parámetros de entrada a la clase formulario? ¿Y sobrecargar el constructor? ¿Y declarar nuevos campos/propiedades?****Se puede sobrecargar el constructor, pasarle parámetros, básicamente y en resumen se puede hacer todo lo que un objeto de clase común puede hacer, ya que en definitiva es eso un objeto de clase.*
63. ***¿Cuál es la diferencia entre Show() y ShowDialog()?****El método Show() visualiza el formulario, ShowDialog() muestra un cuadro de dialogo modal en la aplicación haciendo obligatoria la interacción con este para poder volver al form principal.*
64. ***¿Qué es un formulario MDI? ¿Con que propiedad indico que un formulario es un contenedor MDI? ¿Con qué propiedad del formulario hijo indico cuál es el formulario MDI padre?****MDI son formularios de interfaz de múltiples documentos. Se puede establecer un formulario como MDI mediante la propiedad isMdiContainer = true;*
65. ***Explique el ciclo de vida de los formularios asociándolo a sus eventos correspondientes.****New-> se crea la instancia del formulario.  
    Load-> El formulario está en memoria pero invisible.  
    Paint-> Se dibuja el formulario y sus controles.  
    Activated-> El formulario recibe foco.  
    FormClosing-> permite cancelar el cierre.  
    FormClosed-> el formulario ya es invisible.  
    disposed -> el objeto está siendo destruido.*
66. ***¿Qué es un array “jagged”? ¿En qué valor se inicializan sus elementos?****Es un array anidado, un Array de Arrays y se inicializa en null.*
67. ***¿Los arrays son objetos?****Si.*
68. ***¿Qué significa que en C# los arrays son de “base-cero”?****Que sea base-cero significa que un array con ‘n’ elementos sus índices van de 0 a n-1*
69. ***¿Los arrays implementan la interfaz iEnumerable, ¿Qué es lo que esto les permite hacer?****Esto le permite iterar y recorrer un Array con un Foreach.*
70. ***¿Cómo se declara un array multidimensional?****[acceso] tipo [,] identificador = new tipo[FILAS,COLUMNAS];*
71. ***Consideran que la propiedad Lenght devuelve la cantidad TOTAL de elemtnso de TODAS las dimensiones del array, ¿qué valor vamos a mostrar por consola en el siguiente código?:  
    a. Int[ , , ] a = new int[ 3, 2, 3];  
     int [ , ] b = new int [ 5, 4];  
    Console.WriteLine(“Array lenght = {0}”, a.Lenght + b.Lenght);****Devolvería 3\*2\*3 + 5\*4 = 38.*
72. ***Es lo mismo declarar una variable string o String.****Si porque la palabra “string” es un alias de String.*
73. ***¿Se puede recorrer un String con foreach?****Si, ya que es un objeto.*
74. ***¿Qué significa que los string son inmutables? ¿Qué sucede en realidad cuando usamos métodos u operadores para modificar un string?****Que son inmutables quiere decir que son de solo lectura (readonly). Los modificadores o métodos lo que hacen es devolver una nueva cadena de string y no modifican la ya creada.*
75. ***¿Cuál es la diferencia entre las colecciones y las matrices?****La principal diferencia es que las colecciones pueden agrandarse y achicarse a medida que cambian las necesidades de la aplicación.*
76. ***¿Cuál es la diferencia entre las colecciones genéricas y las no genéricas?****Una colección genérica cumple la seguridad de tipos para que ningún otro tipo de datos se pueda agregar. Las no genéricas almacenan el dato como tipo Object.*
77. ***¿Es necesario determinar el tipo de dato o realizar una conversión al recuperar un objeto de una colección genérica?****No.*
78. ***Describa los siguientes tipos de colecciones genéricas: Dictionary, List, SortedList.****Dictionary: Representa una colección de pares de clave y valor que se organizan por clave.  
    List: Representa una lista de objetos que pueden ser obtenidos mediante un índice. Proporciona métodos para buscar, ordenar y modificar listas.  
    SortedList: Representa una colección de pares de clave y valor que se ordenan por claves según la implementación de interfaz ICompare<T> asociada.*
79. ***¿Qué son y cuál es la diferencia entre una cola (queue) y una pila(Stack)? Asocie con los conceptos “FIFO” y “LIFO”.****Son dos colecciones muy similares que varian la forma en que guardan y extraen los elementos.  
    Stack: Todo nuevo elemento ingresa al final de la misma, y únicamente es posible extraer el último elemento de la colección. LIFO(Last Input Firts Ouput).  
    Cola: Todo nuevo elemento se agrega al principio de la colección y solo se puede extraer el último elemento. FIFO (First Input First Outpu)*
80. ***Describa los siguientes tipos de colecciones no genéricas: ArrayList, Hashtable.****ArrayList: Representa una matriz de objetos cuyo tamaño aumenta dinámicamente según sea necesario.  
    Hashtable: Representa una colección de pares de clave y valor que se organizan por código hash de la clave.*
81. ***¿Se pueden ordenar directamente las colas y las pilas? ¿Por qué? ¿Alternativa?****No, por el método que emplean para extraer y agregar elementos. La alternativa es pasarlo a lista, ordenarlos y pasarlos nuevamente a Stack o Queue.*
82. ***¿Cuál es la diferencia entre las colas y pilas genéricas y las colas y pilas no genéricas?****La diferencia es como guardan los objetos uno los guardan del tipo Object, y las otras lo guardan de un tipo de dato especifico.*
83. ***¿Qué muestra el código?****Opción C.*
84. ***¿Qué es y para que sirve una propiedad?****Una propiedad es un miembro que proporciona un mecanismo flexible para leer, escribir o calcular el valor de un campo. Permiten que una clase exponga de una manera pública una forma de obtener y establecer valores, a la vez que se oculta el código de implementación o validación*
85. ***¿Para que sirve el descriptor de acceso “get”?****Sirve para devolver el valor.*
86. ***¿Para que sirve el descriptor de acceso “set”? ¿Cuál es el papel de la palabra clave “value”?****Set se usa para asignar un nuevo valor y la palabra clave “value” se usa para definir el valor que va a asignar el descriptor de acceso set.*
87. ***¿Cómo declaro una propiedad de solo lectura?****Para que sea de solo lectura tiene que tener un descriptor de acceso get pero no set*
88. ***¿Cómo declaro una propiedad de solo escritura?****Para que sea de solo escritura tiene que tener un descriptor de acceso set pero no get.*
89. ***¿Qué es un enumerado? ¿Cuál es su función?****Son un conjunto propio de constantes con nombres. Permiten declarar un conjunto de nombres u otros valores literales que definen todos los valores posibles que se pueden asignar a una variable. Por dentro estas constantes están asociadas con el tipo de dato int.*
90. ***¿Un enumerado solo puede estar anidado dentro de una clase?****Es mejor definir un enum directamente dentro de un espacio de nombres para que todas las clases del espacio puedan acceder a él. Pero puede estar anidado dentro de una clase o struct.*
91. ***¿Cuál es el primer valor númerico de un enumerado por defecto? ¿Se pueden sobreescribir los valores por defecto?****El primer valor por defecto es 0, y si se pueden sobreescribir esos valores.*
92. ***Indique los valores asociados a cada constante:****Sat 0, Sun 1, Mon 15, Tue 16, Wed 17, Thu 2, Fri 3.*
93. ***¿Qué significa indexar?****Ordenar una serie de datos o informaciones de acuerdo a un criterio común a todos ellos.*
94. ***¿Qué permite un indexador?****Un indexador permite permite indexar las instancias de una clase como matrices.*
95. ***¿Cuál es la diferencia a la hora de declarar un indexador y una propiedad?****La diferencia es que en sus descriptores de acceso usan parámetros y que se utiliza la palabra this para su definición.*
96. ***¿Los indexadores solo se pueden indexar por valores numéricos?****No.*
97. ***¿Un indexador puede recibir más de un parámetro?****Si.*
98. ***¿Los indexadores pueden ser sobrecargados?****Si, se puede ya que cambia la firma del método.*
99. ***¿Cuál es el papel de la palabra clave “this” en un indexador?****Se usa para definir los indexadores.*
100. ***¿Cuál es el papel de la palabra clave “value” en un indexador?****Se usa para definir el valor asignado por el indexador set.*
101. ***¿Se pueden declarar indexadores estáticos? ¿Por qué?****No, ya que hace referencia a objetos instanciados y cuando se usa static no puede usarse la palabra this.  
     Ejemplo de indexadores:  
     public tipoRetorno this[ parametro ]  
     {  
      get { return palabras[i]; (o lógica necesaria) }  
      set { palabras[i] = value; }  
     }*
102. ***Defina encapsulamiento.****Es el ocultamiento del estado, es decir, de los datos miembro de un objeto de manera que solo se pueda cambiar mediante las operaciones definidas para ese objeto. Se encarga de mantener ocultos los procesos internos que necesita para hacer lo que sea que haga, dándole al programador acceso solo a lo que necesita.*
103. ***Defina cada nivel de ocultamiento.****Público: Todos pueden acceder a los datos o métodos de una clase que se definen con este nivel, es lo que tú quieres que la parte externa vea.  
     Protegido: Podemos decir que estas no son de acceso público, solamente son accesibles dentro de su clase y herencias.  
     Privado: es accesible solo por la propia clase.  
     Internal: Es accesible por todo el proyecto.*
104. ***¿Qué es la herencia y su propósito?****Es una relación entre clases en la cual una clase comparte la estructura y comportamiento definido a la otra clase. El propósito es el organizar mejor las clases que componen una determinada realidad, y poder agruparlas en función de atributos y comportamientos comunes. A la vez que cada una se especializa según sus particularidades.*
105. ***¿Qué nombre recibe la clase que hereda y la heredada?****La clase en la que está basada la nueva clase es la clase base o padre, mientras que la clase que hereda se conoce como clase hija, o derivada.*
106. ***¿Qué significa que la herencia es transitiva?****Significa que si C hereda de B y B hereda de A, entonces C también hereda de A.*
107. ***¿Se heredan los constructores?****No, se hereda todo menos constructores y finalizadores.*
108. ***¿Se heredan los miembros private de la clase base?****Si se heredan, aunque no sean visibles.*
109. ***¿Qué es herencia múltiple? ¿Es posible en C#? ¿En que se diferencia con la simple?*** *Herencia Simple: Una clase derivada solo puede heredar de una clase base. Esto se utiliza en C#.  
     Herencia Múltiple: Una clase derivada puede heredar de una o más clases.*
110. ***¿Una clase publica puede heredar de una clase privada?****No, una clase derivada no puede ser más accesible que su clase base.*
111. ***¿Qué es una clase sealed?****Son clases “selladas” que no pueden heredar, es decir no se puede heredar de ella, pero si pueden ser derivadas.*
112. ***¿Una clase sellada puede heredar de otras clases?****Si, si puede.*
113. ***¿Cómo actúa el modificador “protected” en los miembros de la clase base para una clase derivada y como para una clase no-derivada?.****El modificador protected funciona como público para las clases derivadas, y para las no-derivadas como un privado.*
114. ***¿Qué pasa si la clase derivada no hace una llamada explícita a un constructor de la clase base? ¿Qué pasa si la clase base declaro explícitamente un constructor con parámetros de entrada?  
     ?????????????????????????????????***
115. ***¿Qué es el polimorfismo en el contexto de la programación orientada a objetos?****Es la propiedad que tienen los objetos de permitir invocar genéricamente un método cuya implementación será delegada al objeto correspondiente recién en tiempo de ejecución, es decir es la capacidad de tratar objetos diferentes de la misma forma.*
116. ***¿Qué implica el polimorfismo basado en herencia?****Implica la definición de métodos en una clase base y sobreescribirlos con nuevas implementaciones en las clases deviradas.*
117. ***¿Dónde reside la definición del método a sobreescribir? ¿Qué palabra clave se usa para definirlo?****Reside en la clase base o padre, se utiliza virtual o abstract.*
118. ***¿Dónde reside la implementación del método a sobreescribir? ¿Qué palabra se usa para implementarlo?****Reside en la clase hija o derivada, y se utiliza override.*
119. ***¿Cuán se resuelve la invocación?****Se resuelve en tiempo de ejecución.*
120. ***Llenar tabla.***
121. ***Si quiero declarar un método que pueda ser sobreescrito en las clases derivadas, ¿qué modificador debo usar?****Abstract o virtual.*
122. ***Si quiero declarar un método que deba ser sobreescrito en las clases derivadas, ¿Qué modificador debo usar?****Override.*
123. ***¿Qué es una clase abstracta? ¿Cuál es su función?****Indica que una clase esta incompleta y que solo se va a utilizar como una clase base.*
124. ***Las clases no-abstractas que derivan de una clase abstracta, ¿Deben implementar todos su métodos abstractos?****Si.*
125. ***Las clases abstractas que derivan de una clase abstracta, ¿deben implementar todos sus métodos abstractos?****No.*
126. ***¿Se pueden declarar miembros abstractos en clases no-abstractas?****No, pero si virtual.*
127. ***¿Para sobreescribir un método se debe heredar de una clase abstracta?****No sé xdxd.*
128. ***Llenar tabla.***