[Módulo - 01] C# y .NET

**(001) -** ¿Qué es el CLR?

El CLR o Common Language Runtime es la parte de .NET encargada de ejecutar las aplicaciones desarrolladas para la plataforma. El funcionamiento del CLR no es trivial, trabaja encima del sistema operativo para aislar a la plataforma de éste. Su funcionamiento es muy parecido, para entendernos, al hipervisor de una máquina virtual. Esto le permite ejecutar aplicaciones .NET multiplataforma.

Adicionalmente, el CLR se encarga también de gestionar la vida de los objetos, declaraciones y recursos a lo largo de la ejecución de una aplicación .NET. La posibilidad de ejecutar varios procesos simultáneos dentro de una aplicación .NET es una tarea de la que se encarga también el CLR.

**(002) -** Describa el proceso de compilación de C#.

In .NET, programs are not compiled into executable files; they are compiled into Microsoft Intermediate Language (MSIL) files, which the CLR then executes. The MSIL (often shortened to IL) files that C# produces are identical to the IL files that other .NET languages produce; the platform is language-agnostic. A key fact about the CLR is that it is common; the same runtime supports development in C# as well as in VB.NET.

C# code is compiled into IL when you build your project. The IL is saved in a file on disk. When you run your program, the IL is compiled again, using the Just In Time (JIT) compiler (a process often called JIT’ing). The result is machine code, executed by the machine’s processor.

**(003) -** ¿Qué es el CTS?

Common Type System (CTS) describes the datatypes that can be used by managed code. CTS defines how these types are declared, used and managed in the runtime. It facilitates cross-language integration, type safety, and high-performance code execution. The rules defined in CTS can be used to define your own classes and values.

OR we can also understand like,

CTS deals with the data type. So here we have several languages and each and every language has its own data type and one language data type cannot be understandable by other languages but .NET Framework language can understand all the data types.

**(004) -** ¿Qué es un tipo por referencia (Reference Type)? ¿En qué se diferencia de un

tipo por valor (Value Type)?

Un tipo por referencia es aquel que almacena la dirección de memoria de un objeto. Un tipo por valor es aquel que almacena el valor de objeto.

**(005) -** Explique las diferencias entre variables escalares y no escalares.

Las variables escalares son aquellas que contienen un dato atómico, unidimensional e indivisible. Las variables no escalares son aquellas que pueden almacenar en su estructura más de un valor (ej.: listas, arrays.)

**(006) -** ¿Cuál es el punto de entrada (entry point) para los programas en C#?

El punto de entradas para las aplicaciones de C# es el método Main

**(007) -** ¿Cuál es la diferencia entre una conversión implícita y una explícita?

En una conversión explícita debo indicar a qué tipo de dato estoy convirtiendo con la notación entre paréntesis. En la conversión implícita, el programa entiende sin necesidad de aclaración por parte del programador que debe convertir la variable actual al tipo de dato donde se lo está intentando asignar.

[Módulo - 02] Programación orientada a objetos

**(008) -** ¿Qué propone el paradigma orientado a objetos? ¿Qué es un paradigma?

La programación orientada a objetos nos propone modelar en un programa informática objetos que existen en el mundo físico, simulando la interacción entre los mismos a través de métodos, y sus característica a través de atributos.

La palabra paradigma se utiliza como sinónimo de “modelo”. Indica de qué manera estaremos entendiendo el programa que queremos realizar; en este caso, realizaremos un modelaje intentando respetar las características de los objetos a modelar.

**(009) -** Nombre a los pilares del paradigma orientado a objetos.

Herencia, polimorfismo, abstracción y encapsulamiento.

**(010) -** ¿Qué es una clase?

Una clase es un tipo de dato que existe y a partir del cual se crearán objetos en el programa. Los mismos pueden ser creados según la necesidad del usuario.

**(011) -** ¿Qué es un objeto? ¿En qué se relacionan con las clases?

Un objeto es la instanciación de una clase. La clase a partir de la cual se instancie el objeto determinará la naturaleza del mismo.

**(012) -** ¿Qué significa instanciar un objeto?

Instanciar un objeto significa crear un nuevo ejemplar de una clase determinada.

**(013) -** Explique qué es la abstracción en el contexto de programación orientada a

objetos y cuál es su relación con las clases.

Abstraction is one of the [key concepts](https://stackify.com/oops-concepts-in-java/) of object-oriented programming (OOP) languages. Its main goal is to handle complexity by hiding unnecessary details from the user. That enables the user to implement more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden complexity.

**(014) -** ¿Qué son los atributos o estado de un objeto?

Los atributos son las características de un objeto, las variables que definen las cosas que un objeto es o no es, y en las cuales se almacena información sobre su estado o su naturaleza.

**(015) -** ¿Qué son los métodos de un objeto?

Los métodos de un objeto son las acciones que ese objeto tiene la capacidad de realizar.

[Módulo - 03] Miembros estáticos y de instancia:

**(016) -** ¿Qué es un método estático? ¿En qué se diferencia de los métodos de

instancia (no estáticos)?

Los métodos estáticos son aquellos que pueden llevarse a cabo desde una clase sin necesidad de realizar una instanciación de un objeto de la misma. La diferencia con los métodos de instancia es que para ejecutar el mismo necesito obligatoriamente la referencia al objeto que lo ejecuta.

**(017) -** ¿Qué es una clase estática? ¿En qué se diferencia de las clases no estáticas?

Las clases estáticas son aquellas para las cuales no necesito una referencia a un objeto para utilizar los métodos o acceder a los atributos/propiedades de la misma. Las misma son creadas por el CLR al inicio del programa, o bien en su primera invocación durante la ejecución.

La principal diferencia con las clases no estáticas es que las estáticas no pueden ser instanciadas. Además, las clases estáticas “vivirán” durante toda la ejecución del programa, y cualquiera cambio que se haga a cualquier atributo de la misma se verá reflejado desde cualquier otra parte del programa, ya que siempre se referirá a la misma instancia.

**(018) -** ¿Puedo tener miembros estáticos en clases no-estáticas? ¿Puedo tener

miembros no-estáticos en clases estáticas?

Puedo tener miembros estáticos en clases no estáticas. Estos miembros (métodos o atributos) podrán ser accedidos a través de la clase, exista o no una instancia de un objeto de esa clase.

No puedo tener miembros no estáticos en clases estáticas, porque necesitaría una referencia a un objeto para poder utilizar/acceder a los mismos, y siendo estática la clase, ésta nunca se instanciará.

**(019) -** ¿Necesito instanciar un objeto de la clase para llamar a sus métodos estáticos?

No, no es necesario.

**(020) -** ¿Cuántos valores distintos pueden existir para un campo estático y cuántos de

uno no-estático cuando se crearon 5 instancias de la clase?

Para un campo estática, solo existirá un valor posible, almacenado por su variable. En el caso de uno no estática, puedo tener 5 valores diferentes en las 5 instancias creadas, y modificar independientemente cada una sin afectar a las demás.

**(021) -** ¿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? ¿Se

puede acceder a miembros estáticos desde un método de instancia?

No, no se puede usar ya que “this” refiere a la instancia actual de un objeto, y en el caso de una clase estática eso no sería válido.

No, no puedo acceder a miembros no estáticos desde un método estático ya que para esto necesitaría una referencia a un objeto. La única excepción sería pasarle como parámetro a dicho método estático un objeto de la misma clase.

Se puede acceder a miembros estáticos desde un método de instancia.

**(022) -** ¿Se pueden declarar variables estáticas dentro de un método (locales)?

No, no es posible.

**(023) -** De dos ejemplos de métodos estáticos que pertenezcan a las clases de .NET

Framework.

Console y Math

**(024) -** De un ejemplo de un método de instancia que pertenezca a las clases de .NET

Framework.

Random

[Módulo - 04] Namespaces y directivas:

**(025) -** ¿Qué es un namespace y cuál es su función principal?

Namespaces are used to provide a "named space" in which your application resides. They're used especially to provide the C# compiler a context for all the named information in your program, such as variable names. Without namespaces, for example, you wouldn't be able to make a class named Console, as .NET already uses one in its System namespace. The purpose of namespaces is to solve this problem, and release thousands of names defined in the .NET Framework for your applications to use, along with making it so your application doesn't occupy names for other applications, if your application is intended to be used in conjunction with another. So namespaces exist to resolve ambiguities a compiler wouldn't otherwise be able to do.

**(026) -** ¿Puedo tener distintos namespaces dentro de un mismo proyecto o

ensamblado de .NET?

Sí, puedo tener mas de un namespace

**(027) -** ¿Para qué se usa la directiva using?

La directiva using se utiliza para poder incluir la referencia a un namespace de un diferente assembly o proyecto en el proyecto actual

**(028) -** ¿Para qué se usa la directiva alias?

Se usa para abreviar el nombre de un namespace

**(029) -** ¿Puedo declarar dos clases independientes/distintas con el mismo

identificador dentro del mismo namespace? ¿Y en namespaces distintos?

No es posible declarar dos clases con el mismo identificador dentro de un mismo namespace, ya que el compilador no sabrá a cual se refiere cuando sean utilizadas.

Es posible tener dos clases con el mismo nombre en namespaces distintos

[Módulo - 05] Objetos:

**(030) -** ¿Los objetos se crean en tiempo de diseño, de compilación o de ejecución?

Los objetos se crean en tiempo de ejecución.

**(031) -** ¿Cuántos objetos de tipo Alumno puedo tener en mi sistema? ¿Cuántas clases

Alumno tengo en mi sistema? (Dentro de un mismo namespace).

Puedo tener tantos objetos de tipo Alumno como sea necesario, pero solo una clase Alumno por namespace.

**(032) -** Explique el ciclo de vida de un objeto. Detalle las funciones del operador new,

del constructor y del Garbage Collector.

In the .Net Framework objects have the following life cycle:

Object creation (new keyword, dynamic instantiation or activation, etc).

The first time around, all static object initializers are called.

The runtime allocates memory for the object in the managed heap.

The object is used by the application. Members (Properties/Methods/Fields) of the object type are called and used to change the object.

If the developer decided to add disposing conditions, then the object is disposed. This happens by coding a using statement or manually calling to the object’s Dispose method for IDisposable objects.

If the object has a finalizer, the GC puts the object in the finalization queue.

If the object was put in the finalization queue, the GC will, at an arbitraty moment in time, call the object’s finalizer.

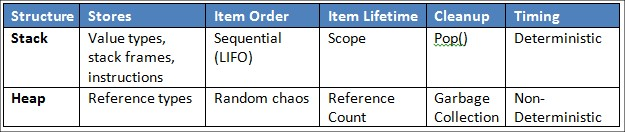
Object is destroyed by marking its memory section in the heap segment as a Free Object.

The CLR Garbage Collector intervenes in the most critical steps in the object lifecycle; GC is almost completely managing steps 3, 6, 7 and 8 in the life of an object.

**(033) -** ¿En qué segmento de memoria se almacenan los tipos valor (value type) y en

cuál los tipos por referencia (reference type)? ¿En cuál interfiere al Garbage

Collector?



**(034) -** Compare y describa: Destrucción determinista y no determinista. Asocie con

el concepto de variables y objetos.

La destrucción determinista es aquella en la que el momento de destrucción del objeto o de liberación de memoria de la variable sucede en un punto en el tiempo determinado. Puede ser realizado por el usuario.

En cambio, en la destrucción no determinista no se tiene noción de cuándo sucede ya que lo decide el Garbage Collector.

**(035) -** ¿Cuál es la diferencia entre declarar, inicializar e instanciar un objeto?

Declare means to tell the compiler that something exists, so that space may be allocated for it. This is separate from defining or initializing something in that it does not necessarily say what "value" the thing has, only that it exists. In C/C++ there is a strong distinction between declaring and defining. In C# there is much less of a distinction, though the terms can still be used similarly.

Instantiate literally means "to create an instance of". In programming, this generally means to create an instance of an object (generally on "the heap"). This is done via the new keyword in most languages. ie: new object();. Most of the time you will also save a reference to the object. ie: object myObject = new object();.

Initialize means to give an initial value to. In some languages, if you don't initialize a variable it will have arbitrary (dirty/garbage) data in it. In C# it is actually a compile-time error to read from an uninitialized variable.

Assigning is simply the storing of one value to a variable. x = 5 assigns the value 5 to the variable x. In some languages, assignment cannot be combined with declaration, but in C# it can be: int x = 5;.

[Módulo - 06] Constantes:

**(036) -** ¿Qué es una constante? ¿Se puede declarar constantes estáticas (static const)?

Una constante es una variable de cualquier tipo que será asignada con un valor en su declaración, y dicho valor no variará durante toda la ejecución del progama.

No puede declararse static a una constante.

**(037) -** ¿Cuándo se asigna el valor a las constantes (tiempo de compilación o

ejecución)? ¿Dos objetos del mismo tipo pueden tener distintos valores en una

misma constante? Relacione con atributos de instancia y estáticos.

La misma se asigna en tiempo de compilación. No es posible modificarla durante la ejecución.

Dos objetos de un mismo tipo no pueden tener valores distintos en una constante, ya que la misma se asigna en la declaración, la cual es común a todos los objetos.

[Módulo - 07] Constructores:

**(038) -** ¿Qué es un constructor? ¿Cuál es su función?

Un constructor es un método que tiene como objetivo instanciar un nuevo objeto del tipo de la clase a la que pertenece, y devolver su referencia para ser asignada a una variable. El mismo se encargará de asignar los atributos de la clase con los valores recibidos, o con valores por defecto.

**(039) -** ¿Qué es el constructor por defecto? ¿Qué sucede con el mismo cuando

declaramos un constructor nuevo en la clase?

El constructor por defecto es un constructor implícito que es implementado en todas las clases propias de manera automática. El mismo es público, no recibe parámetros y se encarga de inicializar todos los atributos con valores por defecto (“” para strings, 0 para números, null para objetos y false para booleanos).

El mismo dejará de existir una vez que se declare un constructor propio.

**(040) -** ¿Con qué valores se cargan los atributos cuando se llama al constructor por

defecto?

“” para strings, 0 para numeros, null para objetos y false para booleanos.

**(041) -** ¿Qué es y para qué sirve un constructor estático? ¿En qué se diferencia su

sintaxis de los constructores de instancia?

Un constructor estático será utilizado por el CLR al lanzar el programa o la primera vez que se utilice una clase estatica. El mismo servirá para asignar los valores designados a los atributos de dicha clase estática. La sintaxis es la misma que en los constructores de instancia, excepto que no podrá modificarse su accesibilidad (siempre son privados), no podrá recibir parámetros, y será ejecutado sólo una vez. Además deberá llevar el modificador static.

**(042) -** ¿Cuántas veces se puede llamar a un constructor estático? ¿Quién lo puede

llamar?

Solo una vez. Lo llamará solo el CLR.

**(043) -** ¿Se ejecutará primero un constructor estático o uno de instancia?

Se ejecutará primero uno estático.

**(044) -** ¿Qué significa sobrecargar un método o constructor?

Significa redefinir un método ya existente modificando su firma (es decir, la cantidad y/o el tipo de parámetros que recibe). De esta manera se le permite al usuario contar con un mismo método que realiza diferentes funciones según la cantidad de parametros que recibe.

**(045) -** ¿Qué debe cambiar para que la sobrecarga de un método o constructor sea

válida?

Debe cambiar su firma. En la misma deberá variarse el tipo de parametros que recibe, y/o su cantidad.

**(046) -** ¿La sobrecarga se resuelve en tiempo de ejecución o en tiempo de

compilación? ¿Cómo se distingue a qué sobrecarga llamar?

El mismo será resuelto en tiempo de compilación, ya que tiene la característica de ser constante durante todo el programa. Se distingue por la cantidad y tipo de parámetros con los cuales se hace la llamada.

**(047) -** ¿Se tiene en cuenta el nombre o identificador de los parámetros de entrada

para una sobrecarga?

No, no se tiene en cuenta.

**(048) -** ¿Se tiene en cuenta el modificador de visibilidad para una sobrecarga?

No, no se tiene en cuenta.

[Módulo - 09] Sobrecarga de métodos:

**(049) -** ¿Los métodos pueden tener el mismo nombre que otros elementos de una

misma clase? (atributos, propiedades, etc).

Sí, es posible.

**(050) -** Mencione dos razones por las cuales se sobrecargan los métodos.

1: Para poder contar con más de un método dentro de una clase que realicen acciones diferentes pero que tengan el mismo nombre

2: Para poder repetir una funcionalidad base que varié según la cantidad de parámetros que reciba, agregando funcionalidades a ese método “base”.

**(051) -** ¿Los métodos estáticos pueden ser sobrecargados?

Sí, de la misma manera que los de instancia.

**(052) -** ¿Agregar el modificador “static” sin cambiar los parámetros de entrada es una

sobrecarga válida?

No, no lo es, ya que la firma sigue siendo la misma

**(053) -** ¿Agregar un modificador “out” o “ref” en la firma del método sin cambiar nada

más es una sobrecarga válida?

Sí, lo es

**(054) -** ¿Cambiar el tipo de retorno sin cambiar los parámetros de entrada es una

sobrecarga válida?

No, no lo es, ya que el compilador no sabrá a cual de los dos métodos llamar al tener más de un método que recibe la misma cantidad y tipo de parámetros.

**(055) -** Si tenemos distintas sobrecargas de un método, ¿cómo podemos reutilizar

Código?

Podemos hacer que de todos estos métodos, siempre y cuando sean compatible, el que reciba la mayor cantidad de informacion llame a el/los que tienen menos parámetros en su firma.

[Módulo - 10] Sobrecarga de constructores:

**(056) -** ¿Para qué se utiliza el operador “this()”?

Se utiliza para llamar a un constructor de la misma clase desde otro constructor sobrecargado.

**(057) -** ¿Se pueden sobrecargar los constructores estáticos?

No, no se pueden sobrecargar porque no reciben parámetros.

**(058) -** ¿Se puede llamar a un constructor estático con el operador “this()”?

No, ya que el operador this() hace referencia a un constructor de la instancia actual del objeto.

**(059) -** ¿Se puede llamar a constructores de otras clases con el operador “this()”?

No, no es posible.

**(060) -** ¿Se puede sobrecargar un constructor privado?

Sí, es posible, de la misma forma en que sobrecarga cualquier otro método.

[Módulo - 11] Sobrecarga de operadores:

**(061) -** ¿Qué es un operador? ¿En qué se diferencian un operador unario y un

operador binario? De un ejemplo de cada uno.

Un operador es carácter reservado por el lenguaje que opera sobre los parámetros recibidos de una manera específica. Un operador unario opera sobre un parámetro (Ej. ‘!’), mientras que un binario sobre dos (‘+’ o ‘-’).

**(062) -** ¿Qué varía en la sintaxis de la sobrecarga de operadores unarios y binarios?

La cantidad de parámetros que reciben en la firma.

**(063) -** ¿Se pueden sobrecargar los operadores de operación y asignación (+=, -=, \*=,

/=)? ¿Por qué?

No, porque estaría sobrecargando la funcionalidad del ‘=’ utilizado para asignar valores, sin lo cual el programa no podría funcionar.

**(064) -** ¿Cuál es la diferencia entre un operador de conversión implícito y uno

explícito? (En finalidad, declaración y aplicación)

La diferencia principal es que la conversión explícita se escribe el tipo de dato al que quiero convertir antes de la asignación, mientras que en la implícita simplemente lo asigno.

**(065) -** Los operadores de casteo “(T)x” no se pueden sobrecargar. ¿Cuál es la

alternativa?

-----

**(066) -** ¿Cuál es la diferencia entre castear (casting), convertir (converting) y parsear

(parsing)?

Explicit conversions (casts): Explicit conversions require a cast operator. Casting is required when information might be lost in the conversion, or when the conversion might not succeed for other reasons. Typical examples include numeric conversion to a type that has less precision or a smaller range, and conversion of a base-class instance to a derived class

Implicit conversions: No special syntax is required because the conversion is type safe and no data will be lost. Examples include conversions from smaller to larger integral types, and conversions from derived classes to base classes.

User-defined conversions: User-defined conversions are performed by special methods that you can define to enable explicit and implicit conversions between custom types that do not have a base class–derived class relationship.

Parsing: siempre desde un tipo hacia String.

**(067) -** ¿Los formularios son objetos?

Sí, todos derivan de object.

**(068) -** ¿De qué clase heredan todos los formularios?

System.Windows.Form

**(069) -** ¿Qué es una partial class o clase parcial?

Una clase cuyo contenido (método y atributos) está dividido en más de un archivo. De esta manera se logra mayor claridad de código.

**(070) -** ¿Puedo agregar parámetros de entrada a la clase del formulario? ¿Y

sobrecargar el constructor? ¿Y declarar nuevos campos/propiedades?

Sí a las tres.

**(071) -** ¿Cuál es la diferencia entre Show() y ShowDialog()?

Show mostrará el formulario, y ShowDialog lo mostrará de forma Modal, es decir, tomando el foco de la pantalla y haciendo posible que el usuario sólo interactúe con éste.

**(072) -** ¿Qué es un formulario MDI? ¿Con qué propiedad indico que un formulario es

un contenedor MDI? ¿Con qué propiedad del formulario hijo indico cuál es el

formulario MDI padre?

A Multiple Document Interface (MDI) programs can display multiple child windows inside them. This is in contrast to single document interface (SDI) applications, which can manipulate only one document at a time. Visual Studio Environment is an example of Multiple Document Interface (MDI) and notepad is an example of an SDI application. MDI applications often have a Window menu item with submenus for switching between windows or documents.

IsMdiContainer = true;

frm2.MdiParent = this;

**(073) -** Explique el ciclo de vida de los formularios asociándolo a sus eventos

correspondientes.

The events in the lifecycle of a Form from the time it is launched to the time it is closed are listed below:

Move: This event occurs when the form is moved. Although by default, when a form is instantiated and launched, the user does not move it, yet this event is triggered before the Load event occurs.

Load: This event occurs before a form is displayed for the first time.

VisibleChanged: This event occurs when the Visible property value changes.

Activated: This event occurs when the form is activated in code or by the user.

Shown: This event occurs whenever the form is first displayed.

Paint: This event occurs when the control is redrawn.

Deactivate: This event occurs when the form loses focus and is not the active form.

Closing: This event occurs when the form is closing.

Closed: This event occurs when the form is being closed.

[Módulo - 13] Arrays:

**(074) -** ¿Qué es un array “jagged”? ¿En qué valor se inicializan sus elementos?

Un array formado de otros arrays. En null.

**(075) -** ¿Los arrays son objetos?

Sí

**(076) -** ¿Qué significa que en C# los arrays son de “base-cero”?

Se refiere a su indexación. El primer elemento del mismo estará en la posición 0.

**(077) -** Los arrays implementan la interfaz IEnumerable, ¿qué es lo que esto les

permite hacer?

Permite recorrer a los arrays con un foreach

**(078) -** ¿Cómo se declara e instancia un array multidimensional?

Int[,] array=new int[10,10];

**(079) -** Considerando que la propiedad Length devuelve la cantidad TOTAL de

elementos 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 Length = {0}”, a.Length + b.Length);

Imprimirá 38. A.Length es igual a 18, y B.Length es igual a 20.

[Módulo - 14] Strings:

**(080) -** ¿Es lo mismo declarar una variable como string (en minúscula) o como String

(con la primera letra en mayúscula)? ¿Por qué?

Sí, string en minúscula es un alias de la clase String.

**(081) -** ¿Se puede recorrer un string con un foreach? ¿Por qué?

Sí, porque la clase String implementa la interfaz IEnumerable, y porque a su vez es un conjunto de Chars

**(082) -** ¿Qué significa que los strings son inmutables? ¿qué sucede en realidad cuando

usamos métodos u operadores para modificar un string?

Significa que una instancia de un objeto tipo string no puede modificarse. Los caracteres que lo componen son de sólo lectura. Cuando operamos sobre un objeto tipo string, se está creando una copia del string existente adicionando esa modificación.

[Módulo - 15] Colecciones:

**(083) -** ¿Cuál es la diferencia entre las colecciones y las matrices?

**(084) -** ¿Cuál es la diferencia entre las colecciones genéricas y las no genéricas?

**(085) -** ¿Es necesario determinar el tipo de dato o realizar una conversión al recuperar

un objeto de una colección genérica?

**(086) -** Describa los siguientes tipos de colecciones genéricas: Dictionary, List,

SortedList.

**(087) -** ¿Qué son y cuál es la diferencia entre una cola (queue) y una pila (stack)?

Asocie con los conceptos “FIFO” y “LIFO”.

**(088) -** Describa los siguientes tipos de colecciones no genéricas: ArrayList, Hashtable.

**(089) -** ¿Se pueden ordenar directamente las colas y las pilas? ¿Por qué (piense en la

función de dichas colecciones)? ¿Cuál es la alternativa?

**(090) -** ¿Cuál es la diferencia entre las colas y pilas genéricas y las colas y pilas no

Genéricas?

**(091) -** ¿Qué muestra el siguiente código?

c. 8, 6, 4, 2, 0,

[Módulo - 16] Propiedades:

**(092) -** ¿Qué es y para qué sirve una propiedad?

Una propiedad sirve para encapsular un atributo de una clase y esconderlo del exterior de la misma. A través de esta propiedad puedo devolver el dato, o asignarlo. A ambas acciones puedo implementarle una validación o lógica específica.

**(093) -** ¿Para qué sirve el descriptor de acceso “get”?

Sirve para devolver un dato determinado.

**(094) -** ¿Para qué sirve el descriptor de acceso “set”? ¿Cuál es el papel de la palabra

clave “value”?

Sirve para asignar un dato recibido al atributo especificado. Value se refiere al valor recibido a través del operador de asignación.

**(095) -** ¿Cómo declaro una propiedad de sólo lectura?

Utilizando solo el descriptor de acceso Get.

**(096) -** ¿Cómo declaro una propiedad de sólo escritura?

Utilizando solo el descriptor de acceso Set.

[Módulo - 17] Enumerados:

**(097) -** ¿Qué es un enumerado? ¿Cuál es su función?

**(098) -** ¿Un enumerado sólo puede estar anidado dentro de una clase?

**(099) -** ¿Cuál es el primer valor numérico de un enumerado por defecto? ¿Se pueden

sobrescribir los valores por defecto?

**(100) -** Indique los valores asociados a cada constante:

enum Day {Sat, Sun, Mon=15, Tue, Wed, Thu=2, Fri};

[Módulo - 18] Indexadores:

**(101) -** ¿Qué significa indexar?

Significa darle a una clase la posibilidad de acceder a alguno de sus atributos a través de un índice que puede ser numérico o de cualquier otro tipo.

**(102) -** ¿Qué permite un indexador (función)?

Permite acceder a un dato del objeto a través de un índice definido por el usuario.

**(103) -** ¿Cuál es la diferencia a la hora de declarar un indexador y una propiedad?

public object this[int index]

{

get { /\* return the specified index here \*/ }

set { /\* set the specified index to value here \*/ }

}

**(104) -** ¿Los indexadores solo se pueden indexar por valores numéricos?

No, cualquier tipo de dato es válido.

**(105) -** ¿Un indexador puede recibir más de un parámetro (ser multidimensionales)?

Sí

**(106) -** ¿Los indexadores pueden ser sobrecargados?

Sí

**(107) -** ¿Cuál es el papel de la palabra clave “this” en un indexador?

Se refiere a la instancia actual del objeto a través del cual se utiliza el indexador.

**(108) -** ¿Cuál es el papel de la palabra clave “value” en un indexador?

Es el dato recibido con el descriptor Set para ser asignado en el índice especificado.

**(109) -** ¿Se pueden declarar indexadores estáticos? ¿Por qué?

No, porque debe existir una instancia del objeto para poder accederlo con un indexador.

[Módulo - 19] Encapsulamiento:

**(110) -** Defina encapsulación en el contexto del paradigma

orientado a objetos.

En programación modular, y más específicamente en programación orientada a objetos, se denomina encapsulamiento al 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.

Cada objeto está aislado del exterior, es un módulo natural, y la aplicación entera se reduce a un agregado o rompecabezas de objetos. El aislamiento protege a los datos asociados de un objeto contra su modificación por quien no tenga derecho a acceder a ellos, eliminando efectos secundarios e interacciones.

**(111) -** Defina cada nivel de ocultamiento / accesibilidad de la programación

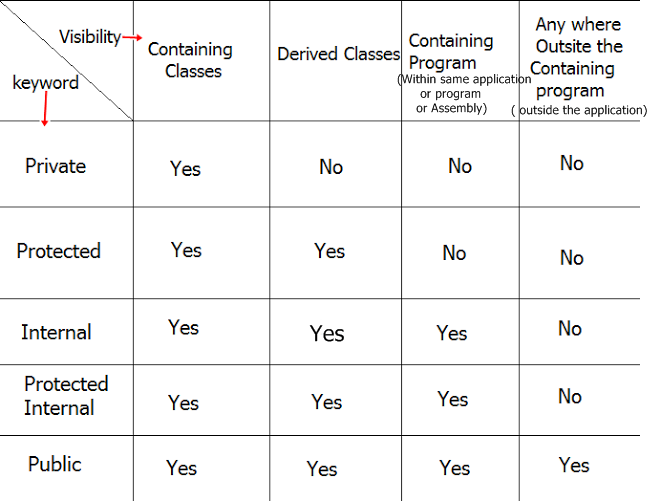
orientada a objetos. Incluya la función del modificador “internal” en C#.

Public: puede ser accedido desde cualquier parte del código.

Private: puede ser accedido sólo desde esa misma clase.

Protected: sólo puede ser accedido por esa clase y sus derivadas.

Internal: sólo desde es mismo programa.



[Módulo - 20] Herencia:

**(112) -** ¿Qué es la herencia en el contexto de la programación orientada a objetos?

¿Cuál es su propósito?

Es una propiedad que permite que los objetos sean creados a partir de otros ya existentes, obteniendo características (métodos y atributos) similares a los ya existentes. Es la relación entre una clase general y otra clase más específica. Es un mecanismo que nos permite crear clases derivadas a partir de clase base, Nos permite compartir automáticamente métodos y datos entre clases subclases y objetos.

**(113) -** ¿Qué nombre recibe la clase que hereda y qué nombre recibe la clase que es

heredada?

Clase hija y clase padre, o derivada y base.

**(114) -** ¿Qué significa que la herencia es transitiva?

Una clase derivada sólo puede tener una clase base directa, pero la herencia es transitiva. Si ClaseC se deriva de ClaseB y ClaseB se deriva de ClaseA, ClaseC hereda los miembros declarados en ClaseB y ClaseA.

**(115) -** ¿Se heredan los constructores?

Sí

**(116) -** ¿Se heredan los miembros private de la clase base?

No

**(117) -** ¿Qué es herencia múltiple? ¿Es posible en C#? ¿En qué se diferencia de la

herencia simple?

Únicamente hay herencia múltiple de interfaz, pero no de clase. Esto implica que puedes heredar las declaraciones, pero no las implementaciones.

**(118) -** ¿Una clase pública puede heredar de una clase privada?

No, porque una será menos accesible que la otra.

**(119) -** ¿Qué es una clase sellada (sealed)?

Una clase que no podrá ser base. Es decir, ninguna clase podrá hereda de ésta.

**(120) -** ¿Una clase sellada puede heredar de otras clases? (Ser clase derivada)

Sí, puede.

**(121) -** ¿Cómo actúa el modificador “protected” en los miembros de la clase base para

una clase derivada y cómo para una clase no-derivada? Relacionar la respuesta con

los modificadores “public” y “private”.

Para una clase derivada, protected es equivalente a public. Para una clase no derivada, protected es equivalente a private.

**(122) -** ¿Qué pasa si la clase derivada no hace una llamada explícita a un constructor

de la clase base? En esta situación, ¿qué pasa si la clase base declaró explícitamente

un constructor con parámetros de entrada?

Si no se explicitaron constructores, no dará error, ya que automáticamente el constructor por defecto de la clase hija llamará al constructor por defecto de la clase base. Si en la clase base se explicitaron parámetros de entrada para su constructor, entonces dejará de existir el constructor automático por defecto y será necesario en la clase derivada llamar al constructor de la clase base.

[Módulo - 21] Polimorfismo:

**(123) -** ¿Qué es el polimorfismo en el contexto de la programación orientada a

objetos?

Concepto:En programación orientada a objetos se denomina polimorfismo a la capacidad que tienen los objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación. Un objeto polimórfico es una entidad que puede contener valores de diferentes tipos durante la ejecución del programa.

En algunos lenguajes, el término polimorfismo es también conocido como ‘Sobrecarga de parámetros’ ya que las características de los objetos permiten aceptar distintos parámetros para un mismo método (diferentes implementaciones) generalmente con comportamientos distintos e independientes para cada una de ellas.

**(124) -** ¿Qué implica el polimorfismo basado en herencia?

Implica que una clase derivada puede sobreescribir un método implementado o declarado en la clase base.

[Módulo - 22] Sobreescritura de métodos (Override):

**(125) -** ¿Dónde reside la definición del método a sobrescribir? ¿Qué palabra clave se

usa para definirlo?

La definición reside en la clase base o en una interfaz, la redefinición en la clase derivada. Se utiliza la palabra reservada Override.

**(126) -** ¿Dónde reside la implementación del método a sobrescribir? ¿Qué palabra

clave se usa para implementarlo?

La definición reside en la clase base, la redefinición en la clase derivada. Se utiliza la palabra reservada Override.

**(127) -** ¿Cuándo se resuelve la invocación? (Tiempo de ejecución o compilación)

Compilación

**(128) -** ¿Cuáles son las diferencias entre sobrecargar (overload) y sobrescribir

(override) un método? (Llenar la tabla)

|  |  |  |
| --- | --- | --- |
| Criterio | Overload | Override |
| Firma (Diferencias o no diferencias en las firmas) | Diferencia necesaria en la firma | No hay diferencia en la firma |
| Ubicación (Misma clase / Clases diferentes) | Se sobrecarga un método ya existente en esa clase | Se sobreescribe un método de una clase base |
| Tiempo de resolución (Compilación / Ejecución) | Compilación | Ejecución |
| Tipo objeto / Tipo Referencia (¿Qué determina  cuál implementación se utilizará?) | Tipo de objeto | Tipo de referencia |

[Módulo - 23] Clases y miembros abstractos:

**(129) -** Si quiero declarar un método que pueda ser sobrescrito en las clases derivadas,

¿qué modificador debo usar?

Virtual

**(130) -** Si quiero declarar un método que deba ser sobrescrito en las clases derivadas,

¿qué modificador debo usar?

Abstract

**(131) -** ¿Qué es una clase abstracta? ¿Cuál es su función?

Una clase que no podrá ser instanciada. Su función es servir como base para otras clases que necesiten derivar de ésta.

**(132) -** Las clases no-abstractas que derivan de una clase abstracta, ¿deben

implementar todos sus métodos abstractos?

Sí, deben implementarlos obligatoriamente.

**(133) -** Las clases abstractas que derivan de una clase abstracta, ¿deben implementar

todos sus métodos abstractos?

No, no es necesario que los implementen.

**(134) -** ¿Se pueden declarar miembros abstractos en clases no-abstractas?

No, no es posible, ya que una clase no abstracta podrá ser instanciada, lo cual no admitiría que un método sea abstracto.

**(135) -** ¿Para sobrescribir un método se debe heredar de una clase abstracta?

No. Un método que proviene de una clase base no abstracta también puede ser sobreescrito (si tiene el modificador virtual).

**(136) -** Llenar los campos de la siguiente tabla con SÍ o NO según corresponda.

|  |  |  |  |
| --- | --- | --- | --- |
| Tipo de Clase | Puede heredar de  otras clases (ser  derivada) | Puede heredarse  de ella (ser base) | Puede ser  instanciada |
| normal (sin  modificadores) | Sí | Sí | Sí |
| abstract | Sí | Sí | No |
| sealed | Sí | No | Sí |
| static | No | No | No |