1. ¿Qué es una excepción? ¿Qué es el manejo de excepciones (exception handling)?  
   Las excepciones son un error en el programa, estás detienen el flujo actual del programa y si no se hace nada el programa dejará de funcionar. El programador debe habilitar su programa para que resuelva estos problemas sin bloquearse. El manejo de excepciones son la forma en que el programador interviene o “atrapa” estos errores, en C# la forma de hacerlo es con los bloques try-catch, la idea consiste en separar físicamente las instrucciones básicas del programa para el flujo de control normal (Try), de las instrucciones para tratamiento de errores (catch).
2. ¿Qué son las excepciones en C#? ¿Qué tienen en común todas las excepciones?  
   Las excepciones en C# son la forma de gestionar errores. Lo que tienen en común es que todas derivan de de la clase Exception, que es parte del runtime de lengauje común (CLR).
3. ¿Qué es el Stack Trace o pila de llamadas? ¿En qué orden se lee?  
   Es todo el hilo de excepciones que se fueron produciendo. Desde la ultima hasta la primera que se produjo.
4. ¿Cómo capturo una excepción? ¿Cuál es la función de cada bloque?   
   Respondida en la 1.
5. En el caso de una estructura try-catch se tiene más de un tipo de bloque catch, ¿se podría ejecutar más de un bloque catch que forme parte de la misma estructura?  
   Si, capturando la excepción de un tipo en el catch y lanzar una nueva excepción. (creo)
6. ¿En qué parte del código continúa la ejecución del programa una vez manejada una excepción?  
   En el primer bloque catch que encuentre el runtime.
7. ¿Existe una forma de capturar cualquier excepción sin importar su tipo? ¿Qué habría que considerar si se tiene más de un tipo de bloque catch?  
   Si, se puede tener una excepción general y ésta tendría que ir al final de todos los catch pertenecientes al bloque Try.
8. ¿Qué sucede cuando se lanza una excepción? ¿Qué sucede si no la manejo/controlo?   
   En caso de producirse la excepción, el runtime detiene la ejecución normal y empieza a buscar un bloque **catch** que pueda capturar la excepción pendiente (basándose en su tipo). Si no manejas o controlas la excepción el programa se cerrará.
9. ¿Cómo lanzo una excepción?  
   Con la palabra reservada Throw.
10. Dentro de un bloque catch, ¿cuál es la diferencia entre “throw;” y “throw ex;” (ex es un identificador para una excepción capturada)?  
    Throw ex lanza una excepción ya capturada, el otro no lanza nada xD.
11. ¿Se lanzan en tiempo de compilación o de ejecución? ¿Por qué?  
    Se lanzan en tiempo de ejecución ya que estos errores son en tiempo de ejecución.
12. ¿Cómo creo una excepción propia?  
    Derivándola de la clase Exception.
13. ¿Qué es la propiedad InnerException? Describa a qué clase pertenece, su contenido y cómo se carga. ¿Qué sucede si no se le proporciona un valor?   
    Pertenece a la clase Exception, su contenido es una referencia a una excepción anterior, se carga pasandole en el constructor la excepción capturada a una nueva excepción.
14. Bloque finally: ¿En qué condiciones se ejecutará el código que contiene? ¿Cómo se ubica dentro de una estructura de manejo de excepciones? ¿Para qué es útil?  
    El bloque **finally** es útil en dos casos: para evitar la repetición de instrucciones y para liberar recursos tras el lanzamiento de una excepción. En general, el código dentro de un bloque finally se ejecuta cuando el control sale de un bloque try como resultado de la ejecución normal, "break", "throw", "continue" o "return" o debido a una excepción de la declaración try.
15. Indique el orden y describa brevemente qué sucede en cada etapa del ciclo de vida general de los sistemas. (mantenimiento – diseño – desarrollo/construcción – implementación – análisis – pruebas)  
    No sé.
16. ¿Qué es una prueba unitaria?  
    Son casos de prueba para cada función no trivial o método en el módulo, de forma que cada caso sea independiente del resto.
17. ¿Qué es una prueba integral?  
    Pruebas integrales son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias y lo que prueban es que todos los elementos unitarios que componen el software, funcionan juntos correctamente probándolos en grupo.
18. ¿Qué es una prueba funcional?   
    Una prueba funcional es una prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software.
19. ¿Qué es el patrón AAA? Describa cada una de sus etapas.   
    El patrón AAA (Arrange, Act, Assert) es una forma habitual de escribir pruebas unitarias para un método en prueba.  
    La sección Arrange de un método de prueba unitaria inicializa objetos y establece el valor de los datos que se pasa al método en pruebas.  
    La sección Act invoca al método en pruebas con los parámetros organizados.  
    La sección Assert comprueba si la acción del método en pruebas se comporta de la forma prevista.
20. ¿Qué es la clase Assert? ¿Para qué sirve?  
    Explícita para determinar si el método de prueba se supera o no.  
    Cumple su tarea a través de métodos estáticos.  
    Estos métodos analizan una condición True – False.
21. ¿Qué es una clase genérica? ¿Qué permite?   
    Es una clase parametrizada que es exactamente igual a una clase de las habituales, salvo por un pequeño detalle: su definición contiene algún elemento que depende de un parámetro que debe ser especificado en el momento de la declaración de un objeto de dicha clase. Lo que permite tipar dicha clase.
22. ¿Se puede tener más de un parámetro genérico en una clase o un método?  
    Si
23. ¿Qué es una restricción o constraint?  
    Es una forma de acortar la posibilidad de datos que se le pueden pasar a la clase o método genérico y así controlar el tipo de tipado que va a tener.
24. ¿Qué sucede si no hay restricciones?   
    Acepta cualquier tipo de dato.
25. ¿Qué sucede si intentamos instanciar una clase genérica pasando como argumento un tipo que no cumple con las restricciones?  
    Error en tiempo de diseño.
26. ¿Puedo declarar métodos genéricos en clases no-genéricas?  
    Si.
27. ¿Sólo se puede aplicar una sola restricción por parámetro?   
    No.
28. ¿Qué nombre pueden tener los comodines o parámetros genéricos?  
    Cualquier nombre pero se suelen poner letras mayúsculas, ej: T, U, V, Z, etc.
29. Complete la siguiente tabla:

Restricción Descripción

where T : struct El argumento debe ser un tipo de valor

where T : class El argumento de tipo debe ser un tipo de referencia

where T : notnull El argumento de tipo no debe ser un tipo de referencia y no debe contener ningún miembro de tipo de referencia.

where T : new() El argumento debe tener un constructor sin parámetros público. Y se especifica en último lugar

where T : <clase base > El argumento debe ser o derivarse de la clase especificada.

where T : <interface > El argumento de tipo debe ser o implementar la interfaz especificada.

where T : U El argumento de tipo proporcionado por T debe ser o derivarse del argumento proporcionado por U.

**INTERFACES.**

(030) ¿Qué es una interfaz?  
Es un “contrato” que establece una clase en la cual asegura que implementara un conjunto de métodos. Son una manera de describir que debería hacer una clase sin especificar el cómo. Es la descripción de uno o más métodos que posteriormente alguna clase pueda implementar.

(031) ¿Qué puede especificar una interfaz (atributos, métodos, propiedades, etc)?   
Permite especificar métodos y propiedades.

(032) ¿Qué nivel de visibilidad/acceso pueden tener los miembros especificados?   
Son públicos y no se permite especificarlo. Son parecidos a los “abstractos” ya que no cuentan con implementación.

(033) ¿Las clases sólo pueden implementar una interfaz?   
No, las clases pueden implementar varias interfaces.

(034) ¿Se puede elegir cuáles de las operaciones definidas en la interfaz implementar en una clase?   
No, ya que se debe implementar toda la interfaz.

(035) ¿Una clase puede heredar de otra y al mismo tiempo implementar una o más interfaces? Describa la sintaxis para lograr esto.   
primero se indica la clase base y luego la interfaz separada de una coma CLASE : CLASEBASE, INTERFAZ.

(036) Si implementé una interfaz en la clase base, ¿debo implementarla también en las derivadas?   
No.

(037) ¿Qué significa implementar una interfaz de forma explícita? ¿Qué utilidad tiene? ¿Qué consecuencias o efectos tiene?  
 Sirven para ocultar la implementación de miembros de interfaces a las clases que lo implementan, y también sirve para evitar ambigüedad. La consecuencia es que las clases que derivan de clases que implementan interfaces de manera explícita no pueden sobreescribir los métodos definidos explícitamente.

(038) ¿Una interfaz puede implementar otra interfaz? ¿Qué sucede con las operaciones especificadas (se suman o se anulan)?   
No sé

(039) ¿Se pueden tener interfaces genéricas? ¿Se puede restringir sus parámetros de tipo? ¿Puede implementarlas cualquier clase o sólo clases que también sean genéricas?

Si puede ser genérica y con restricciones, puede implementarla cualquier clase.

(040) ¿En qué se diferencian una interfaz y una clase abstracta?   
que sus métodos y propiedades no tienen modificador de visibilidad.

(041) Si un método tiene un parámetro del tipo de una interfaz:

a) ¿Qué argumentos podré pasarle?

El mismo tipo de interfaz o clases que la implementen.

b) Si no casteo el argumento a otro tipo, ¿podré acceder a todos los métodos y

propiedades del objeto?

No, solo a los que implementa de la interfaz.

c) Considerando que una interfaz no tiene implementación, ¿si llamo a uno de sus

métodos qué implementación se ejecutará? Asocie con polimorfismo.

Ejecutará la implementación del método más específica.

(042) ¿Qué es serializar?   
Es el proceso de convertir un objeto en memoria en una secuencia lineal de bytes.

(043) ¿Para qué sirve serializar?   
Para pasarlo a otro proceso, otra máquina, grabarlo en disco, grabarlo en una base de datos.

(044) ¿En qué formatos se puede serializar (vistos en clase)? Indique qué miembros de la clase se incluyen en cada formato.  
Serialización XML:  
. Solo serealiza los atributos y propiedades públicas.  
. La clase central es XmlSerializer y sus métodos más importante Serialize y Deserialize  
. Crea archivos C# (.cs) y los compila en archivos .dll en el directorio especificado; la serialización se produce con esos archivos .dll  
SERIALIZACIÓN BINARIA:  
. Serializa y deserializa objetos en formato binario.  
. Puede serializar atributos públicos y privados.  
. La clase es BinaryFormatter()

(045) Indique qué características debe tener la clase para ser serializable en cada formato.  
Debe tener arriba del nombre de la clase [Serializable] y un constructor por defecto, en el caso de Xml este tendrá que ser público.

**BASES DE DATOS:**

(046) ¿Qué es una base de datos?   
Una **base de datos** es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. Es un conjunto de tablas con diferentes tipos de datos.

(047) ¿Qué es una tabla?   
Las tablas son objetos de base de datos que contienen todos sus datos. En las tablas, los datos se organizan con arreglo a un formato de filas y columnas, similar al de una hoja de cálculo. Cada fila representa un registro único y cada columna un campo dentro del registro.

(048) ¿Qué es una primary key o clave primaria?   
se llama **clave principal** a un campo o a una combinación de campos que identifica de forma única a cada [fila](https://es.wikipedia.org/wiki/Registro_(base_de_datos)) de una [tabla](https://es.wikipedia.org/wiki/Tabla_(base_de_datos)).

(049) ¿Para qué sirve la clase SQLConnection?   
Para conectarse con la base de datos.

(050) ¿Para qué sirve la clase SQLCommand?  
 Sirve para interactuar de diferentes formas con la base de datos.

(051) ¿Qué es una “connection string” o cadena de conexión? ¿Qué clase de ADO.NET la utiliza?   
Es donde se especificaran los datos (usuario,servidor,etc) de una conexión a una fuente de datos. La utiliza la clase SQLConnection.

(052) ¿Qué es la “Inyección SQL”? ¿Cómo se puede evitar con ADO.NET (tecnología utilizada en clase para conectar a una base de datos)?   
**Inyección SQL** es un método de infiltración de código intruso que se vale de una [vulnerabilidad informática](https://es.wikipedia.org/wiki/Error_de_software) presente en una aplicación en el nivel de validación de las entradas para realizar operaciones sobre una [base de datos](https://es.wikipedia.org/wiki/Base_de_datos).

(053) Explique las 2 formas vistas en clase de liberar los recursos de conexión incluso cuando ocurra una excepción. Escriba un ejemplo de cada unaNo sé xD.

(054) En base a la siguiente tabla: HACER xD

(055) ¿Qué es un delegado? ¿Para qué sirve?   
Un [delegado](https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/builtin-types/reference-types) es un tipo que representa referencias a métodos con una lista de parámetros determinada y un tipo de valor devuelto. Cuando se crea una instancia de un delegado, puede asociar su instancia a cualquier método mediante una signatura compatible y un tipo de valor devuelto. Puede invocar (o llamar) al método a través de la instancia del delegado. Sirve para encapsular un método de modo que se pueda llamar de forma anónima.

(056) ¿Qué papel juega el tipo de retorno y los parámetros en la declaración del delegado?   
Es la firma del delegado y también de los métodos que podrán ser llamados o utilizados mediante ese delegado.

(057) ¿En qué se diferencian los delegados de los punteros a función vistos en el lenguaje C?  
La diferencia es que un delegado solo puede “apuntar” a métodos con la misma firma, tienen seguridad de tipos.

**THREADS**:

(058) ¿Qué es un hilo o subproceso?   
es una secuencia de tareas encadenadas muy pequeña que puede ser ejecutada por un sistema operativo. Es simplemente una tarea que puede ser ejecutada al mismo tiempo que otra tarea.

(059) ¿Qué es un proceso?   
Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son en conjunto conocidos como un proceso.

(060) ¿Cuáles son los posibles parámetros de entrada para el constructor de la clase Thread? ¿Qué son? ¿Para qué sirven? ¿En qué se diferencian?   
Los posibles parámetros son un método, y ParameterizedThreadStart que permitirá pasarle parámetros al método que le estamos pasando al Hilo.

(061) ¿Cuándo termina la vida de un hilo? Explique las distintas posibilidades.  
Termina cuando el método pasado por parámetro finaliza. O cuando lo abortas.

(062) Explique el siguiente código línea a línea e indique para qué se utiliza: HACER.

**EVENTOS:**

(063) ¿Qué es un evento?   
Es el modo que tiene una clase en particular de proporcionar notificaciones a sus clientes cuando ocurre algo en particular dentro del objeto. Es un mensaje enviado por un objeto para indicar que se ha producido una acción invocada programáticamente o por un usuario.

(064) ¿Qué es un manejador de un evento (event handler)?   
Es el procedimiento que captura el evento. Es básicamente un método con la misma firma del delegado que se ejecuta cuando se dispara el evento.

(065) ¿Por qué los eventos son de un tipo delegado? ¿Cómo impacta esto a los posibles manejadores de ese evento?   
Ya que es necesario tener un componente que enlace el emisor del evento con el receptor del evento e impacta de forma en que al evento solo se le pueden asociar manejadores(métodos) con la misma firma del delegado.

(066) ¿Qué sintaxis se utiliza para subscribirse a un evento? ¿Y para desubscribirse?   
+= manejador o -=manejador.

(067) ¿Un mismo manejador puede estar subscripto a distintos eventos simultáneamente?   
No sé. Yo creo que mientras tenga la misma firma del delegado si.

(068) ¿Un mismo evento puede tener más de un manejador distinto?   
Si.

(069) ¿Qué papel juega la clase emisora y cuál la clase receptora? Asocie con los conceptos de declarar el evento, lanzar el evento y subscribirse al evento.   
la clase emisora produce el evento de un tipo de delegado el cual tipificara la firma del manejador, cuando se lanza el evento se busca el manejador o manejadores con los que está asociado y estos se ejecutan.

(070) ¿Puedo declarar eventos estáticos?   
Si.

(071) ¿Puedo definir un evento en una interfaz?   
Si.

(072) Si tengo un evento de instancia y 5 instancias de esa clase. ¿Cuántas veces voy a tener que asociar el evento al manejador en la clase receptora?  
No sé xD

(073) ¿Qué es un método de extensión? ¿Para qué sirve?   
Permite “agregar” métodos a los tipos existentes sin crear un nuevo tipo derivado, recompilar o modificar de otra manera el tipo original

(074) ¿Qué características debe tener la clase que lo contiene?   
La clase debe ser static y los métodos también.

(075) ¿Qué características debe tener el método para ser considerado un método de extensión?   
Tiene que ser static y además como parámetro se pasara la palabra reservada this [Tipo de dato al cual se va a extender] [nombre]

(076) ¿Cómo se usa un método de extensión? Indique diferencias entre su declaración y su uso.   
Se declara como un método static pero se utiliza como un método de instancia en la clase extendida.

(077) ¿Puedo usar un método de extensión sin referenciar previamente al namespace donde está declarado (con un “using”, por ejemplo)?  
Los métodos de extensión únicamente se encuentran dentro del ámbito cuando el espacio de nombres se exporta explícitamente en el código fuente con una directiva using.