CUESTIONARIO PARTE 2

**Módulo 24 – Excepciones**

¿Qué es una excepción? ¿Qué es el manejo de excepciones (exception handling)?

Cuando se produce un error se lanza una excepción. Las excepciones detienen el flujo actual del programa, y si no se hace nada, el programa dejará de funcionar.

**(002)** ¿Qué son las excepciones en C#? ¿Qué tienen en común todas las excepciones?

Todas las excepciones son objetos que derivan de la clase Exception,, que es parte del runtime de lenguaje común (CLR).

**(003)** ¿Qué es el Stack Trace o pila de llamadas? ¿En qué orden se lee?

El stack es una pila de datos, uno encima del otro . Se lee last in , first out (LIFO)

**(004)** ¿Cómo capturo una excepción? ¿Cuál es la función de cada bloque?

Una excepción se caputra un un bloque Try – Catch

Try -> El bloque try contiene una expresión que puede generar la excepción.

Catch -> ejecuta algo si se produjo una excepcion

Throw -> Cuando necesita lanzar una excepción, el runtime ejecuta una instrucción throw y lanza una excepción definida por el sistema, o una propia.

Finally -> La cláusula finally de C# contiene un conjunto de instrucciones que es necesario ejecutar sea cual sea el flujo de control.

Las instrucciones del bloque finally se ejecutarán aunque el control abandone un bucle try como resultado de la ejecución normal porque el flujo de control llega al final del bloque try.

**(005)** 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?

Un bloque de código en una instancia try puede contener muchas instrucciones, cada una de las cuales puede producir una o más clases diferentes de excepción. •Al haber muchas clases de excepciones distintas, es posible que haya muchos bloques catch y que cada uno de ellos capture un tipo específico de excepción.

- Osea , se ejecuta SOLO el catch mas especifico respecto al tipo de excepción.

**(006)** ¿En qué parte del código continúa la ejecución del programa una vez manejada una excepción?

Despues del catch que se haya ejecutado

**(007)** ¿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?

Un bloque catch general (Exception), puede capturar cualquier excepción independientemente de su clase y se utiliza con frecuencia para capturar cualquier posible excepción que se pudiera producir por la falta de un controlador adecuado.

En caso de existir, un bloque catch general debe ser el último bloque catch en el programa.

**(008)** ¿Qué sucede cuando se lanza una excepción? ¿Qué sucede si no la manejo/controlo?

el runtime detiene la ejecución normal y empieza a buscar un bloque catch que pueda capturar la excepción pendiente (basándose en su tipo).

Si en la función inmediata no se encuentra un bloque catch adecuado, el runtime desenreda la pila de llamadas en busca de la función de llamada.

Si tampoco ahí encuentra un bloque catch apropiado, busca la función que llamó a la función de llamada y así sucesivamente hasta encontrar un bloque catch (o hasta llegar al final, en cuyo caso se cerrará el programa).

Osea va hacia atrás buscando un catch , si no lo encuentra , crashea a la mierda

**(009)** ¿Cómo lanzo una excepción?

throw new Exception(X,X);

**(010)** Dentro de un bloque catch, ¿cuál es la diferencia entre “throw;” y “throw ex;” (ex es un identificador para una excepción capturada)?

Throw estaría lanzando una nueva excepción , throw ex estaría relanzando la excepción capturada por ese mismo catch.

**(011)** ¿Se lanzan en tiempo de compilación o de ejecución? ¿Por qué?

De ejecución , porque es ahí cuando se generan.

**(012)** ¿Cómo creo una excepción propia?

Haciendo una clase que herede de Exception.

**(013)** ¿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?

La propiedad InnerException es una exception que se guarda dentro de otra exception . se accede mediando miExceptionX.InnerException = miExepcionY , o mediante miExceptionX = new Exception (“”,miExeptionY), si no , tiene valor null.

**(014)** 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?

Se ejecuta SIEMPRE , independientemente de si el código que estaba en el try corrió correctamente , o si lanzo una exception y se fue para un catch. Va al final de todo del bloque después de los catch . Es útil para evitar repetición de instrucciones y para liberar recursos después de el lanzamiento de una exception.

**Módulo 25 – Pruebas de calidad**

**(015)** 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)

xd

**(016)** ¿Qué es una prueba unitaria?

Casos de prueba para cada función no trivial o método en el módulo, de forma que cada caso sea independiente del resto.

**(017)** ¿Qué es una prueba integral?

Pueban que todos los elementos unitarios que componen el software, funcionan juntos correctamente probándolos en grupo.

**(018)** ¿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.

**(019)** ¿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 pruebas

.•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

**(020)** ¿Qué es la clase Assert? ¿Para qué sirve?

Clase Assert:

•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.

**Módulo 26 – Tipos genéricos**

**(021)** ¿Qué es una clase genérica? ¿Qué permite?

Una clase parametrizada 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.

Permite crear clases genéricas , capaces de implementar un tipado fuerte sin necesidad de conocer a priori los tipos para los que serán utilizadas.

**(022)** ¿Se puede tener más de un parámetro genérico en una clase o un método?

Si señor

**(023)** ¿Qué es una restricción o constraint?

Las restricciones limitan el tipo de objeto que puede recibir la clase.

**(024)** ¿Qué sucede si no hay restricciones?

Le podes tirar cualquier cosa , que capas eso después te trae problemas y excepciones

**(025)** ¿Qué sucede si intentamos instanciar una clase genérica pasando como argumento un tipo que no cumple con las restricciones?

No se puede

**(026)** ¿Puedo declarar métodos genéricos en clases no-genéricas?

YEAH

**(027)** ¿Sólo se puede aplicar una sola restricción por parámetro?

No podes poner varias pero algunas son excluyentes de otras.

**(028)** ¿Qué nombre pueden tener los comodines o parámetros genéricos?

El que quieras , pero normalmente se usa T U

**(029)** Complete la siguiente tabla:

|  |  |
| --- | --- |
| Restriccion | Descripcion |
| where T : struct | El argumento de tipo debe ser un tipo de valor. |
| where T : class | El argumento de tipo debe ser un tipo de referencia. |
| where T : notnull | El argumento no puede ser null |
| where T : new() | El argumento de tipo debe tener un constructor sin parámetros público. Cuando se usa conjuntamente con otras restricciones, la restricción new() debe especificarse en último lugar. |
| where T : <clase base > | El argumento de tipo debe ser o derivarse de la clase base especificada. |
| where T : <interface > | El argumento de tipo debe ser o implementar la interfaz especificada. Pueden especificarse varias restricciones de interfaz. La interfaz de restricciones también puede ser genérica. |
| where T : U | El argumento de tipo proporcionado por T debe ser o derivarse del argumento proporcionado para U. |

**Módulo 27 – Interfaces**

**(030)** ¿Qué es una interfaz?

Es la descripción de uno o más métodos que posteriormente alguna clase puede implementar

**(031)** ¿Qué puede especificar una interfaz (atributos, métodos, propiedades, etc)?

C# no permite especificar atributos en las interfaces.

Todos los métodos son como “abstractos” ya que no cuentan con implementación (no se permite especificarlo).

Se pueden especificar propiedades (sin implementación).

**(032)** ¿Qué nivel de visibilidad/acceso pueden tener los miembros especificados?

Todos los métodos son públicos (no se permite especificarlo).

**(033)** ¿Las clases sólo pueden implementar una interfaz?

Las clases pueden implementar varias interfaces.

**(034)** ¿Se puede elegir cuáles de las operaciones definidas en la interfaz implementar en una clase?

No , tenes que implementar todo.

**(035)** ¿Una clase puede heredar de otra y al mismo tiempo implementar una o más interfaces?

Para implementar una interface a una clase derivada, primero hay que indicar la clase base, luego la interface separadas por una coma (,).

**(036)** ¿Las interfaces se heredan?

Que se yo , se heredan los métodos que aplico la clase base

**(037)** ¿Qué significa implementar una interfaz de forma explícita? ¿Qué utilidad tiene? ¿Qué consecuencias o efectos tiene?

Mediante la Implementación explícita, podes ocultar lo que no usas

Se coloca el nombre de la interfaz adelante del nombre del método que se esta implementando explícitamente y además no se le indica la visibilidad.

Void INombreInterface. MiMetodo ()

{

Console.WriteLine("Hola");

}

Cuando se haga obj.MiMetodo(); tira error como si no existiera, habría que castearlo.o hacer el método implícito.

**Módulo 28 – Archivos y serialización**

**(038)** ¿Qué es serializar?

Es convertir un objeto en memoria en una secuencia lineal de bytes

**(039)** ¿Para qué sirve serializar?

Para pasarlo a otro proceso , maquina , grabarlo en disco , o en una base de datos.

**(040)** ¿En qué formatos se puede serializar (vistos en clase)? Indique qué miembros de la clase se incluyen en cada formato.

En formato XML , que por defecto incluye solo las propiedades y atributos publicos –

En formato binario . Incluye propiedades y atributos públicos y privados.

**(041)** Indique qué características debe tener la clase para ser serializable en cada formato.

En XML la clase debe tener un constructor por defecto , si hay relación de herencia , se debe colocar [XmlIncluede(typeof(Clase))] en la clase base e indicando cada clase heredada.

En Binario se debe agregar [Serializable] , la clase debe tener constructor por defecto ,

**Módulo 29 – 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

**(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?

Es un campo en una tabla que identifica y diferencia a cada columna(objeto) , un id básicamente. No puede ser null

**(049)** ¿Para qué sirve la clase SQLConnection?

Nos permite conectarnos a la base de datos , mediante un connectionString que le especifiquemos.

**(050)** ¿Para qué sirve la clase SQLCommand?

**(051)** ¿Qué es una “connection string” o cadena de conexión? ¿Qué clase de ADO.NET la utiliza?

Es donde se especifican los datos de una connecion a una fuente de datos.

Utiliza System.Data.SqlClient

**(052)** ¿Qué es la “Inyección SQL”? ¿Cómo se puede evitar con ADO.NET (tecnología utilizada en clase para conectar a una base de datos)?

Es cuando el usuario te manda un código en algún lugar donde puede meter datos, y ese código sql lo ejecuta y te cagan todo. Se puede evitar verificando antes los datos ingresados , guardándolos en palabras clave, que luego se transfieren a los datos originales. Todo esto usando SQL Command ,

**En vez de**

using( SqlConnection con = (acquire connection) ) {

con. Open();

using( SqlCommand cmd = new SqlCommand("SELECT \* FROM usuarios WHERE nombre = '" + nombreUsuario + "'", con) ) {

using( SqlDataReader rdr = cmd.ExecuteReader() ){

…

}

}

}

**se podría usar lo siguiente:**

using( SqlConnection con = (acquire connection) ) {

con. Open();

using( SqlCommand cmd = new SqlCommand("SELECT \* FROM usuarios WHERE nombre = @nombreUsuario", con) ) {

cmd.Parameters.AddWithValue("@nombreUsuario", nombreUsuario);

using( SqlDataReader rdr = cmd.ExecuteReader() ){

...

}

}

}

OSEA

COMANDO.PARAMMETERS.ADDWITHVALUE(IDENFICIADOR PARA PARAM, VALOR QUE ASOCIAR)

NONQUERY ES PARA INSERT, UPDATE Y DELETE. PARA NADA MAS

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

Una es el using , como el que puse arriba , la otra es Connection.Close();

**(054)** En base a la siguiente tabla:



a) Escriba las sentencias SQL para insertar las siguientes provincias teniendo en cuenta que “id” es clave primaria y NO es autoincremental:

{ ‘Formosa’, 573.823 }

{‘Neuquen’ } (Sin cantidad de habitantes)

{‘Entre Ríos’, 1.308.000 }

{‘Tierra del Fuego’, 152.317 }

INSERT INTO PROVINCIAS (id,nombre\_provincia,cantidad\_habitantes) VALUES (1,'Formosa',573.823 );

INSERT INTO PROVINCIAS (id,nombre\_provincia) VALUES (2,'Neuquen');

INSERT INTO PROVINCIAS (id,nombre\_provincia,cantidad\_habitantes) VALUES (3,'Entre Rios', 1.308.000 );

b) Teniendo en cuenta las inserciones del punto anterior, modifique el registro de la Provincia ‘Neuquén’ con una nueva cantidad de habitantes de 619.745.

UPDATE PROVINCIAS SET cantidad\_habitantes = 619.745 WHERE Id = 2 ;

c) Escriba la sentencia para consultar a todas las provincias con todas sus columnas.

SELECT \* FROM PROVINCIAS ;

d) Escriba la sentencia para consultar sólo el nombre de todas las provincias.

SELECT nombre\_provincia FROM PROVINCIAS

e) Escriba la sentencia para consultar todos los datos de todas las provincias que tengan más de 600mil habitantes. Dibuje una tabla con el resultado de la consulta (teniendo en cuenta que se ejecutaron las sentencias de los puntos anteriores).

SELECT \* FROM PROVINCIAS WHERE cantidad\_habitantes > 600.000

f) Escriba la sentencia para eliminar el registro con id igual a 3.

DELETE FROM PROVINCIAS WHERE Id = 3 ;

g) Escriba la sentencia para eliminar todos los registros que no tengan datos en la columna “cantidad\_habitantes”.

DELETE FROM PROVINCIAS WHERE cantidad\_habitantes = null;

h) ¿Puedo insertar un registro que no contenga datos para la columna “nombre\_provincia”? ¿Por qué?

No porque no permite null.

**Módulo 30 – Delegados**

**(055)** ¿Qué es un delegado? ¿Para qué sirve?

Un delegate es un tipo que representa referencias a métodos con una lista de parámetros determinada y un tipo de valor devuelto. Es la versión de .NET de los punteros a función de C++ pero con seguridad de tipos (type-safe), es decir, se comprueban los tipos de retorno y de los parámetros del mismo. La clase de delegado no sólo contiene una referencia a un método, también puede mantener referencias a varios métodos mientras cumplan con la firma definida.

Los delegados permiten pasar los métodos como parámetros.

Los delegados pueden usarse para definir métodos callback.

**(056)** ¿Qué papel juega el tipo de retorno y los parámetros en la declaración del delegado

Les da seguridad de tipos, los parametros permiten definir que métodos pueden asociarse a ese delegado , ya que las firmas deben ser iguales.

**(057)** ¿En qué se diferencian los delegados de los punteros a función vistos en el lenguaje C?

En que tienen seguridad de tipos , y que varios métodos se pueden asociar con un delegado.

**Módulo 31 – Hilos**

**(058)** ¿Qué es un hilo o subproceso?

Un hilo es una secuencia de tareas encadenada muy pequeña que puede ser ejecutada por un sistema operativo , una tarea que puede ser ejecutada al mismo tiempo que otra , los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos , son lo que llamamos un proceso.

**(059)** ¿Qué es un proceso?

This.(058);

Un conjunto de hilos

**(060)** ¿Cuáles son los posibles parámetros de entrada para el constructor de la clase Thread? ¿Qué son? ¿Para qué sirven? ¿En qué se diferencian?

Thread x = New Thread(UnMetodo); -> Le mandas un metodo que queres que se vaya ejecutando.Este método no recibe parámetros.

Thread x = New Thread (new ParameterizedThreadStart (Metodo)); ->Le mandas un metodo que recibe parametros . los parametros se pasan cuando haces el x.Start(parametro);

**(061)** ¿Cuándo termina la vida de un hilo? Explique las distintas posibilidades.

-Cuando termina de ejecutar lo que estaba haciendo

-Cuando muere el hilo padre

-Cuando lo detenemos

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



El código se fija si el código se esta ejecutando sobre el hilo principal, si no se necesita hacer un invoke –

Se pasa el método que esta siendo utilizado como un delegado(puntero) ,y los parámetros usados en un array de objects, se invoca al método en el hilo principal ( osea se ejecuta en el principal). La segunda vez , va a entrar en el else.

**Módulo 32 – Eventos**

**(063)** ¿Qué es un evento?

Un evento es un modo que tiene una clase de notificar que sucedió ALGO dentro del objeto.

Cada evento tiene un emisor que produce el evento , y un receptor que lo captura.

**(064)** ¿Qué es un manejador de un evento (event handler)?

El objeto que produce el evento es el emisor, El que lo captura , es el Event handler

**(065)** ¿Por qué los eventos son de un tipo delegado? ¿Cómo impacta esto a los posibles manejadores de ese evento?

Los eventos son de un tipo delegado porque

**(066)** ¿Qué sintaxis se utiliza para subscribirse a un evento? ¿Y para desubscribirse?

objEmisor.evento += MetodoManejador ; y objEmisor.evento -= MetodoManejador ;

**(067)** ¿Un mismo manejador puede estar subscripto a distintos eventos simultáneamente?

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 emisora contiene el evento y el delegado , también los asocia .

La receptora acepta el evento y tiene algún handler , el delegado en la clase emisora invoca el método (event handler) de la clase receptora.

**(070)** ¿Puedo declarar eventos estáticos?

Dentro de una clase estatica 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?

Una por instancia (creo)

**Módulo 33 – Métodos de extensión**

**(073)** ¿Qué es un método de extensión? ¿Para qué sirve?

Es un método que podemos agregar a una clase ya existente , la cual no tenemos acceso.

Sin crear un nuevo tipo derivado , recompilar , o modificar al tipo original.

**(074)** ¿Qué características debe tener la clase que lo contiene?

Es valido para clases y para interfaces , debe ser estatica y no generica.

**(075)** ¿Qué características debe tener el método para ser considerado un método de extensión?

**(076)** ¿Cómo se usa un método de extensión? Indique diferencias entre su declaración y su uso.

Se declaran como estáticos pero se usan como de instancia. Con parámetros this ,

Public static int Metodo(this tipo nombre , tipo nombre , etc)

{ return 0;}

**(077)** ¿Puedo usar un método de extensión sin referenciar previamente al namespace donde está declarado (con un “using”, por ejemplo)?

No , solo aparecen si se importa explícitamente con el using.