**Módulo 24 – Excepciones**

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

Las excepciones detienen el flujo actual del programa, y si no se hace nada, el programa dejará de funcionar. Se producen por un error en el programa. El programa debe construirse usando diferentes técnicas de gestión de errores para ***atrapar*** las excepciones y administrarlas de manera conveniente.

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

En c# la gestión de errores se manejan por medio de excepciones. Todas heredan de Exception.

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

Es la pila de llamadas de excepciones que se fueron capturando, es decir se puede “rastrear” cual fue la excepción que se lanzo primero. Se lee del ultimo al primero.

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

Los bloques **try-catch** son la solución que ofrece la orientación a objetos a los problemas de tratamiento de errores. La idea consiste en separar físicamente las instrucciones básicas del programa para el flujo de control normal de las instrucciones para tratamiento de errores. Así, las partes del código que podrían lanzar excepciones se colocan en un bloque **try**, mientras que el código para tratamiento de excepciones en el bloque **try** se pone en un bloque **catch** aparte.

El bloque **try** contiene una expresión que puede generar la excepción. 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 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). Si encuentra un bloque **catch**, se considera que la excepción ha sido capturada y se reanuda la ejecución normal desde el cuerpo del bloque **catch.**

**Hace que las instrucciones para tratamiento de errores no se mezclen con las instrucciones lógicas básicas, por lo que el programa es más fácil de interpretar.**

(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?

Si se puede. Desde la primer excepción que se podría generar hasta la última… En el caso de que se utilice un tipo de exception GENERAL esta debería ser la última en ser captura, solo se puede tener una general.

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

Si encuentra un bloque **catch**, se considera que la excepción ha sido capturada y se reanuda la ejecución normal desde el cuerpo del bloque **catch.**

(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?

Mirar pregunta 5.

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

Esto interrumpe inmediatamente la secuencia de ejecución normal del programa y transfiere el control al primer bloque **catch** que pueda hacerse cargo de la excepción en función de su clase.

(009) ¿Cómo lanzo una excepción?

Cuando necesita lanzar una excepción, el runtime ejecuta una instrucción **throw** y lanza una excepción definida por el sistema.

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

* throw ex restablece el seguimiento de la pila (por lo que sus errores parecen originarse en HandleException)
* throw no lo hace - el delincuente original sería preservado.

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

Ejecucion.

(012) ¿Cómo creo una excepción propia?

Se debe crear una clase con el nombre de tu excepción, esta clase debe derivar de la clase Exception y por último debo tener un constructor.

(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?

Objeto que describe el error que causó la excepción actual. La propiedad [InnerException](https://docs.microsoft.com/es-es/dotnet/api/system.exception.innerexception?view=netframework-4.8" \l "System_Exception_InnerException) devuelve el mismo valor que se pasó al constructor [Exception(String, Exception)](https://docs.microsoft.com/es-es/dotnet/api/system.exception.-ctor?view=netframework-4.8" \l "System_Exception__ctor_System_String_System_Exception_) o null si no se suministró el valor de la excepción interna al constructor. Esta propiedad es de sólo lectura.

(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 aunque haya o no haya una exception. Se ejecuta al final del catch. Sirve para asegurarse de que antes de la salida se ejecute el bloque de código que esta dentro de finally.

**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)

Diseño, armado, test, aprender.

(016) ¿Qué es una prueba unitaria?

Escribir 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?

Se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión. Pruebas integrales o pruebas de integración 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.

(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.

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?

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.

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

Si, dictionary por ejemplo(¿

(023) ¿Qué es una restricción o constraint?

Una restricción o constraint es por ejemplo where T : string le estas indicando que si bien el valor a recibir es un objeto T digamos, T debe ser un string.

(024) ¿Qué sucede si no hay restricciones?

T puede ser cualquier tipo de object

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

Error en tiempo de compilación(¿

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

Si. Para que un método sea genérico, no hace falta que la clase también lo sea:

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

No. CREO

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

A partir de la letra T por convención.

(029) Complete la siguiente tabla:

|  |  |
| --- | --- |
| Restricción | Descripción |
| Where T : struct | Tipo por valor |
| Where T: class | Tipo referencia |
| Where T : not null | Argumento no nulo. |
| Where T : new () | Constructor sin parámetros público. Cuando se usa conjuntamente con otras restricciones, la restricción new() debe especificarse en último lugar. |
| Where T : <clase base> | Argumento clase base o derivada. |
| 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 un contrato que establece una clase en el cual esta clase asegura que implementará un conjunto de métodos. Son una manera de describir qué debería hacer una clase sin especificar el cómo. Es la descripción de uno o más métodos que posteriormente alguna clase puede implementar.

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

Atributos no, métodos y propiedades sin implementación (públicos y abstractos sin modificadores de acceso) Interfaces si (parecido a herencia)

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

No llevan modificadores de acceso/visibilidad. Son así por defecto (public y abstract)

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

Se pueden implementar varias.

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

No, es un contrato que se firma, se deben implementar todas las operaciones de la interfaz.

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

Si.

(036) ¿Las interfaces se heredan?

No, se implementan.

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

Los miembros implementados explícitamente sirven para ocultar la implementación de miembros de interfaces a las clases que lo implementan. También sirve para evitar la ambigüedad cuando, por ejemplo, una clase implementa dos interfaces las cuales poseen un miembro con la misma firma.

(038) ¿Una interfaz puede implementar otra interfaz? ¿Qué sucede con las operación es especificadas (se suman o se anulan)?

Puede heredarla no implementarla. Se suman.

(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.

Solo las genéricas.

(040) ¿En qué se diferencian una interfaz y una clase abstracta?

las clases abstractas se sobrescriben los métodos ya que tienen virtual con override, en una interfaz no llevan modificadores de acceso y no se

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

a) ¿Qué argumentos podré pasarle?

Los que estén definidos en la interfaz.

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

No.

c) Considerando que una interfaz no tiene implementación, ¿si llamo a uno de sus métodos qué implementación se ejecutará?

Asocie con polimorfismo.

Se ejecutara la implementación que esta definida en la clase.

**Módulo 28 – Archivos y serialización**

(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.

Para pasarlo a otra máquina.

Para grabarlo en disco.

Para 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.

En xml, por defecto incluye sólo las propiedades y atributos públicos.

En binario, Por defecto incluyen todos los atributos y propiedades, ya sean públicas o privadas.

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

La clase XmlSerializer crea archivos C# (.cs) y los compila en archivos .dll en el directorio especificado por la variable de entorno TEMP; la serialización se produce con esos archivos DLL.

Una clase debe tener un constructor por defecto para que XmlSerializer pueda serializarla.

Sólo se pueden serializar los atributos y propiedades públicas.

Los métodos no se pueden serializar.

Para poder hacer una serialización binaria se debe agregar el marcador [Serializable]

Una clase debe tener un constructor por defecto para que BinaryFormatter pueda serializarla.

**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?

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)). Una clave primaria comprende de esta manera una [columna](https://es.wikipedia.org/wiki/Columna_(base_de_datos)) o conjunto de columnas. No puede haber dos filas en una tabla que tengan la misma clave primaria.

(049) ¿Para qué sirve la clase SQLConnection?

La cadena de conexión (Connection String) es donde se especificarán los datos (usuario, servidor, etc.) de una conexión a una fuente de datos.

(050) ¿Para qué sirve la clase SQLCommand?

Representa un procedimiento almacenado o una instrucción de Transact-SQL que se ejecuta en una base de datos de SQL Server. Un comando puede ser de diferentes tipos (Procedimiento Almacenado, etc.), por ahora solo utilizaremos del tipo texto. El comando deberá estar asociado a una conexión, en la cual ejecutará sus acciones.

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

Las cadenas de conexión construidas manualmente en tiempo de ejecución a partir de la entrada del usuario no validada son vulnerables a los ataques de inyección de cadenas y ponen en peligro la seguridad en la fuente de datos. Para solucionar estos problemas, ADO.NET 2.0 introdujo constructores de cadenas de conexión para cada proveedor de datos de .NET Framework . Estos constructores de cadenas de conexión exponen los parámetros como propiedades fuertemente tipadas y hacen posible validar la cadena de conexión antes de enviarla a la fuente de datos.

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

.Close();

using

(054) En base a la siguiente tabla:

|  |  |  |
| --- | --- | --- |
| Colum name | Data type | Allow nulls |
| id | int |  |
| Nombre\_provincia | Nvarchar(50) |  |
| Cantidad\_habitantes | int | √ |

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 }

String consulta;

consulta = "INSERT INTO id++ (name,habitantes)"; VALUES('Formosa',573823) ;

consulta = "INSERT INTO id++ (name,habitantes)"; VALUES('Formosa',573823) ;

comando.CommandText = consulta;

conexion.Open();

comando.ExecuteNonQuery();

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.

String consulta;

consulta = "UPDATE Personas SET cantidad\_habitantes= 619745 WHERE nombre\_provincia= ‘Neuquen’";

comando.CommandText = consulta;

conexion.Open();

comando.ExecuteNonQuery();

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

String consulta;

consulta = "SELECT \*”

comando.CommandText = consulta;

conexion.Open();

comando.ExecuteNonQuery();

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

String consulta;

consulta = "SELECT \* VALUES nombre\_provincia”

comando.CommandText = consulta;

conexion.Open();

comando.ExecuteNonQuery();

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).

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

consulta = "DELETE FROM Personas WHERE id = 3";

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

consulta = "DELETE FROM Personas WHERE cantidad\_habitantes > 1";

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

**Módulo 30 – Delegados**

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

Un delegado es un tipo que representa referencias a métodos con una lista de parámetros determinada y un tipo de valor devuelto.

Un objeto delegado encapsula un método de modo que se pueda llamar de forma anónima. Cuando ocurre el evento, se llama a los delegados que proporcionan los clientes para el evento.

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

Delimita la firma que va a tener el método que se asocie al evento.

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

Los delegados son como los punteros de función de C++, pero tienen seguridad de tipos.

Los delegados permiten pasar los métodos como parámetros.

Los delegados pueden encadenarse entre sí; por ejemplo, se puede llamar a varios métodos en un solo evento.

Un Evento puede tener múltiples manejadores y viceversa

**Módulo 31 – Hilos**

(058) ¿Qué es un hilo o subproceso?

Secuencia de tareas encadenadas muy pequeña que puede ser ejecutada por un sistema operativo. Un hilo 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?

El método utilizado puede tener parámetros.

Para esto deberemos utilizar ParameterizedThreadStart al instanciar el nuevo hilo.

El parámetro se pasará mediante el método Start de dicho hilo.

// Creo el hilo

Thread t = new Thread(new ParameterizedThreadStart(Metodo));

// Inicio el Hilo, indicando el parámetro

t.Start(1000);

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

El proceso sigue en ejecución mientras al menos uno de sus hilos de ejecución siga activo.

En el momento en el que todos los hilos de ejecución finalizan, el proceso no existe más y todos sus recursos son liberados.

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

private void WriteTextSafe (string text)

{

If (textBox1.InvokeRequired)//lo que hace es preguntarle al hilo de ejecucion del texbox si require invocacion para cambiar el txtbox

{

Var d = new SafeCallDelegate(WriteTextSafe);//llama al delegado del metodo writetextsafe

textBox1.Invoke(d, new object [ ] { text } ) ;

}else

{

textBox1.Text = text ;

}

}

**Módulo 32 –Eventos**

(063) ¿Qué es un evento?

Un evento es el modo que tiene una clase dada de proporcionar notificaciones a sus clientes cuando ocurre algo en particular dentro del objeto.

El uso más habitual para los eventos lo vemos en las interfaces gráficas (evento Click de un botón, evento Load de un Form, etc.).

Los eventos proporcionan un medio apropiado para que los objetos puedan señalizar cambios de estado que pueden resultar útiles para los clientes de ese objeto.

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

Es un método.

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

Porque el delegado es el encargado de manejar el evento. El evento en si no tiene valor de retorno ni recibe parámetros, entonces se le asocia a un delegado.

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

objEmisor.evento += MetodoManejador;

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.

Emisora declarar evento y lanzarlo

Receptora suscribirse.

(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?

5 veces, una por instancia.

**Módulo 33 –Métodos de extensión**

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

Son una clase especial de método estático, pero se les llama como si fueran métodos de instancia en el tipo extendido.

Permiten "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?

Debe ser una clase estatica.

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

Debe ser un método estatico en una clase estatica y su primer parámetro debe estar precedido por el modificador this.

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

Se declara como estatico pero se utiliza como si fuera de instancia.

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

Si se puede.