**Excepciones**

* **Gestión de errores**: interceptar errores en tiempo de ejecución, se controlan por medio de excepciones, cuando hay un error se lanza una excepción.
* Una excepción interrumpe el fluyo de un programa, y si no se hace nada este dejara de funcionar. El programa debe estar habilitado a resolver este problema sin bloquearse.
* Las excepciones derivan de la clase exception (que proviene de CLR).
* Ventajas:

-Poseen mensajes de error no representados por enteros y enumerado, son descriptivos y representa errores de forma clara y evidente.

-Cada clase exception va a contener información específica.

* **TRY-CATCH**:
  + TRY: se coloca las instrucciones que pueden lanzar excepciones.
  + CATCH: es donde las excepciones son tratadas.
  + FINALLY: es un bloque de instrucción que siempre se va a ejecutar independiente al flujo de control.
  + Múltiple CATCH: se utiliza cuando una porción de código puede lanzar distintos tipos de excepciones.
  + Se puede crear excepciones particulares. Pero también existen las excepciones genéricas (Catch genéricos), estas pueden capturar cualquier tipo de excepciones, solo pueden usarse una vez por TRY y deben ir como la última excepción.
* **THROW**: lanza una excepción, interrumpiendo la secuencia del programa y lanza una excepción al primer CATCH que pueda hacerse cargo.

**Interfaces**

Es un contrato que establece que una clase va a implementar un conjunto de métodos.

* + Una clase puede implementar varias interfaces.
  + Sus métodos y propiedades no pueden poseer modificadores.
  + No pueden tener atributos.
  + Las interfaces pueden ser públicas.

Utiliza la palabra reservada “interface”. Se implementan mediante herencia, las clases derivadas deben implementar de forma implícita o explícita sus métodos. (No se utiliza override).

**Test Unitario**

Se divide en:

* **Test unitarios**: Son casos de prueba para cada función no trivial o método en el módulo, de forma que cada caso sea independiente.
* **Pruebas de integración**: testea que todos los elementos unitarios funcionen juntos correctamente.
* **Pruebas funcionales**: Son pruebas que se basan en la ejecución, revisión y retroalimentación de las funcionabilidades de un software.

Patrón **AAA** es la forma de diseñar un test.

* + Arrange: Declarar objeto y sus variable.
  + ACT: Llamada a métodos.
  + Assert: Verificar resultados de los métodos.

La clase assert es un clase estática que a partir de sus métodos verifica los resultados de los métodos de una clase, los cuales van a determinar es estado de la prueba.

**Clases Genéricas**

Son **clases parametrizadas**, es decir que hay que especificarles un tipo de valor o un valor a la hora de declararlas.

También se pueden implementar interfaces y métodos genéricos en clases no genéricas pero al igual que una clase genérica que tiene que especificar un parámetro para estas al usarlas(o heredar en caso de las interfaces).

Se puede usar la palabra reservada “**where:**” la cual va a definir una restricción a la hora de declararle parámetro.

**SERIALIZACION**

* **Stream**: Crear archivo de texto.
  + StreamWriter: Crear un archivo de texto en el “path” indicado. Si existe el archivos lo va a sobrescribir y en caso contrario lo va a crear (FALSE). Si existe el archivo va a añadir información y en caso contrario lo va a crear.
    - Write()
    - WriteLine()
    - Close()
  + StreamReader: Lee un archivo en el “path” indicador.
    - Read()
    - ReadToEnd()
    - ReadLine()
    - Close()
* **Serializar**: Es una forma de pasar un objeto o valor a una línea de bytes.
  + SerializarcionXML: Serializa en archivo “.XML”.
    - El tipo de objeto que se le pasa debe tener un constructor por defecto.
    - Permite guardar instancias de objeto. Pero solo sus atributos y propiedades públicos con excepción de las colecciones privadas.
    - No guarda métodos.
      * XmlSerializer(typeof())
      * ser.Serialize(Stream, object)
      * Object ser.Deserialize(stream)
      * XmlTextWriter(Path, encoding)
      * XmlTextReader(Path)
  + Se debe incluir [XmlInclude(Type)] en caso de ser una herencia.
* **Serializacion binaria**
  + Serializa atributos y propiedades privados y públicos
  + BinaryFormatter()
    - Serialize(stream, object)
    - Deserialize(stream)
  + FileStream(path, filemode)

**SQL**

* **SqlConnection**: conexión a la base de datos
  + SqlConnection aux = new SqlConnecction(ConnectionStr);

ConnectionStr= “DataSource = [Instancia del servidor] InicialCatalog= [nombre de la base de datos] IntegratedSecurity=TRUE”;

O

Propieties.Setting.Default.conexion

* **SqlCommand** cmd: permite hacer consultas a la base de datos.
  + SqlCommand cmd = new SqlCommand();

cmd.CommandType = CommandType.Text;

cmd.CommandText= “ ”;

* + - “Insert into [DB]( Nombres de columna) values(‘Pedro’,1)”
    - “Update [DB] set [nombre]=’Pedro’ where id=1”
    - “Delete from[DB] where id=1”

cmd.Connection = aux.Open();

cmd.ExecuteNonQuery();

aux.Close();

* + - “Select\*[nombre de columnas]from[db]”
    - “Select [nombre de comlumna]from[db]where id=1”

Aux.Open();

SqlDataReader reader = cmd.ExecuteNonQuery();

While(reader.Read())

{

Algo=new algo(reader[“columna”][…]);

}

aux.Close();

**THREADS**

Es una **tarea que se ejecuta al mismo tiempo que otra**. Los hilos que comparten recursos y los mismo recursos se llaman procesos.

Un procesos estará activos mientras que un hilo lo este y cuando ya todos los hilos estes inactivos se liberaran sus recursos.

Se puede crear hilos parametrizado con un new ParameterizedThreadStart(), que un thread toma un metodo como atributo.

**EVENTOS**

Un evento es la forma que tiene una clase de notificar al usuario u otra clase que se produjo un cambio u ocurrió algo en el objeto, este también puede ser producido programáticamente o por el usuario.

Este va a estar compuesto por:

Emisor: Método.

Receptor: Manejador de eventos.

Enlazador: Delegado.

Un **delegado** es una especie de puntero a un manejador.

* + Un delegado permite métodos como parámetro.
  + Un evento puede tener múltiples manejador y viceversa.
  + Se puede llamar a varios métodos en un solo evento.