Generics

|  |  |
| --- | --- |
| public class Clase1<T>  **public class Clase1<T,U>**  {  private string nombre;  private T miVarGeneric  } | **Acepta cualquier tipo de dato. Se va reemplazar en todas las T**   * los constructores se declaran normalmente * los gets y set igual |
| public class Clase1<T> where T: class  public class Clase1<T> where T: struct  public class Clase1<T> where T: new()  public class Clase1<T> where T: NombreClase  public class Clase1<T> where T: NombInterfac  public class Clase1<T, U> where T:U | Restrinciones: solo acepta clases(string incluye)  solo acepta int y float  La clase pasada por referencia Al menos debe tener un constructor sin parámetros.  Acepta a esa clase o su derivados(hijos)  “debe implementar la interfaz especifica”  “T debe ser U o derivado |
| public class Clase1<T, U>  where T:Persona, new()  where U:Alumno | Para poner restricciones a varios generics |
| public static bool operator +(Torneo<T> t, Equipo e)  {  if (t != e)  {  t.Equipos.Add((T)e);  }  return true;  } | Hace una sobrecarga de operadores con genercs |

INTERFACES

btnDerechos→agregar→interface

* no llevan modificador de visibilidad y son públicos
* no se definen atributos.
* Las clases que hereden de estas interfaces estan obligadas a implementar:

|  |  |
| --- | --- |
| public interface IMensaje  {  string Nombre { get; }  string Motrar();  int Restar(int num3,int num2);  } | Definir una propiedad  definir un metodo  definir un metodo con parametros. |
| MiClase clase1=new MiClase();  **IMensaje aux=clase1;** | Clase que hereda de IMensaje;  podemos guardar dentro de una variable interface clases que heredan de esa interface.  NOTA: Pero desde aux solo tenemos acceso a las propiedades y metodos que contiene la interface (las que son propieas de la clase no!)(AL IGUAL QUE HERENCIA ) |
| **Interfaces explicitas** | Cuando dos interfaces tiene la misma firma |
| **public interface IInterface1**  {  string Motrar();  }  public interface IInterface2  {  string Motrar();  **}** | Ej interfaces con la misma firma |
| **public class ClasePadre : IInterface1, IInterface2**  {  public string Mostrar()  {  return "General";  }  string IInterface1.Mostrar()  {  return "Desde la Interface 2";  }  string IInterface2.Mostrar()  {  return "Desde la Interface 2";  }  **}** | Clase que implementa las dos interfaces de antes  si no inidicamos nada toma este por defecto(implicitamente)  si casteamos toma este**(no llevan identificador de visibilidad)**  si casteamos toma este**(no llevan identificador de visibilidad)** |
| **USO**  Console.WriteLine(clase1.Mostrar());  Console.WriteLine(((IInterface1)clase1).Mostrar());  Console.WriteLine(((IInterface2)clase1).Mostrar()); | Casteamos a la interface1  Casteamos a la interface2 |

|  |  |
| --- | --- |
| **INTERFACES GENÉRICAS**  **public interface IInterface1<T> where T : class**  {  T Motrar(T tipo);  **}**  **---------------------------------------------------------------------------------------**  **public class ClasePadre<T> : IInterface1<T> where T : class**  **--------------------------------------------------------------------------------------**  **…Uso---**  EL uno es normal como un genérico | Definición de la interfas genérica  la firma de la clase esta obligada a llevar un genérico y tener la restricción de ser unicamente clase(los genéricos) |

TEST UNITARIOS

Para agregar sobre el lugar →btnDerecho→ agregar MSTest Test Proyect (Proyecto de prueba MSTest)

* Agregar la dependencia a nuestro Test
* Agregar el using a nuestro test
* Si es un programa este debe tener el acceso en public. No en internal

|  |  |
| --- | --- |
| public void TestMethod1()  {  //Arrange:Iniciar variables  string nombre1 = "jonathan";  Lista primerLista = new Lista(nombre1);  //Act: Llamar al método a testear  //Assert:comprobar el valor esperado Assert.IsTrue(primerLista.Nombres.Count > 0);  } |  |
| ASSERTS  Assert.IsTrue()  Assert.IsFalse() | Comprueba que retorna true  Comprueba que retorna false |
| [TestMethod] [ExpectedException(typeof(DivideByZeroException))]  public void Test\_Division()  {  //Arrange  string nombre1 = "jonathan";  Lista primerLista = new Lista(nombre1);  //Act  float resul =primerLista.Division(1,0);  } | Para testear una excepción  True si la función arroja esa excepción  no lleva Assert |

Archivos

Ecribir Archivos

*Antes de abrir un streamWriter nuevo se debe cerrar el anterior*

|  |  |
| --- | --- |
| StreamWriter sw = new StreamWriter("miArchivo.txt");  sw.WriteLine("Texto en archivo.");  sw.Close();  sw.Dispose(); | Istanciamos el stream writer definiendo el archivo a abrir (si no existe lo crea)((agregar el parámetro **true** para que continuar))  Escribe el texto en el documento.  Cierra el documento  Libera de la memoria el SW |
| StreamWriter sw = new StreamWriter("miArchivo.txt",true); | Abre el archivo pero escribe al final del documento |
| using(...){  sw.WriteLine("Nuevo texto.");  }; | De esta forma no tenemos que usar close y dispose |

Leer Archivos

*Si no existe el archivo rompe el programa*

|  |  |
| --- | --- |
| try  {  using (StreamReader sr = new StreamReader("miArivo.txt"))  {  Console.WriteLine(sw.ReadLine());  Console.WriteLine(sw.ReadLine());  Console.WriteLine(sw.ReadLine());  };  }  catch(Exception ex)  {  Console.WriteLine(ex.Message);  } | |
| * Al hacer un read, queda el cursor en el final de la lectura * usamos try por que puede romper * using para evitar usar close y dispose. | |
| sr.Read()  sr.ReadLine()  sr.ReadToEnd() | Lee un carácter y retorna un ASCII  lee un linea de texto y retorna un string  lee todo el documento y retorna un string |
| try  {  if (File.Exists("miArchivo.txt"))  {  StreamReader sw = new StreamReader("miArchivo.txt");  Console.WriteLine(sw.ReadToEnd());  sw.Close();  sw.Dispose();  }  else  {  throw new Exception("No se encontro el archivo");  }  }  catch(Exception ex)  {  Console.WriteLine(ex.Message);  } | |
| File.Exists(“nombreArchivo.txt”) | True si el archivo existe; false si no |
| Path.GetFullPath("miArchivo.txt") | Devuelve la ruta completa del archivo  que se encuentra en el mismo lugar que el program |
| Environment.SpecialFolder.AdminTools); Environment.SpecialFolder.Personal); Environment.SpecialFolder.System); Environment.SpecialFolder.MyPictures); Environment.SpecialFolder.Desktop); | Tiene la “direccion” de las carpetas mas comunes |
| Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);  obtenemos la ruta completa del la carpeta especial | |

SERIALIZACION

**Para esto necesitamos que la clase contenga un constructor sin parámetros**

**Necesitamos que los atributos que se van a serializar tengan un getter y setter**

Serializar un objeto en XML

|  |
| --- |
| MiClase objeto1= new MiClase(“jonathan”,12);  using (StreamWriter sw = new StreamWriter("miArchivoXML.xml"))  {  XmlSerializer serializador = new XmlSerializer(typeof(MiClase));  serializador.Serialize(sw, objeto1);  } |

Serializar una lista de objetos en XML

|  |
| --- |
| List<MiClase> listaClases = new List<MiClase>() { objeto1, objeto2, objeto3 };  using (StreamWriter sw = new StreamWriter("miArchivoXML.xml"))  {  XmlSerializer serializador= new XmlSerializer(typeof(List<MiClase>));  serializador.Serialize(sw, listaClases);  } |

Deserilizar un objeto

|  |
| --- |
| **using (StreamReader sr = new StreamReader("miArchivoXML.xml"))**  **{**  **XmlSerializer serializador = new XmlSerializer(typeof(MiClase));**  **MiClase objeto1 = (MiClase)serializador.Deserialize(sr);**  *Console.WriteLine(objeto1.Mostrar());*  **}** |
| Nota: si queremos usar este objeto fuera del using: se debe declarar a fuera del using |
| using (StreamReader sr = new StreamReader("miArchivoXML.xml"))  {  XmlSerializer serializador = new XmlSerializer(typeof(List<Miclase>));  List<Miclase> listaClases = (List<Miclase>)serializador.Deserialize(sr);  foreach(var i in listaClases)  {  Console.WriteLine(i.Mostrar());  }  } |

JSON(transfomar de clase a string y viceversa)

|  |
| --- |
| string stringGuardador = JsonSerializer.Serialize(objeto1);  *si queremos guardarlo podemos hacerlo en un txt* |
| MiClase nuevaClase=JsonSerializer.Deserialize<MiClase>(stringGuardador); |

SQL y VS

1. importar Biblioteca SQLCLient
   1. sobre la solución→administrar paquetes de NuGet→ buscar: “sqlclient”→instalar el System.Data.SqlClient()
2. agregar using System.Data.SqlClient;

|  |  |
| --- | --- |
| string stringConexion = "Server=.;DataBase=MiPrimerBase;Trusted\_Connection=True";  SqlConnection conSql = new SqlConnection(stringConexion);  SqlCommand comando = new SqlCommand();  comando.CommandType=System.Data.CommandType.Text;  comando.Connection = conexionSql;  comando.CommandText = "SELECT \* FROM EMPLEADOS";  conSql.Open();  SqlDataReader sr=comando.ExecuteReader();  sr.Read();  string aux = sr["NOMBRE"].ToString();  conSql.Close();  Console.WriteLine(aux); | String donde guardamos datos de la conexión  itanciamos al conexión con la base de datos  intanciamos la clase de manejo de comandos  definimos que nos manejaremos con string  definimos la conexión con los comandos  definimos la consulta  abrimos la conexión  clase para guardar la respuesta de la consulta  ejecutamos la consulta y la guardamos  guardamos lo que necesitamos en un string  cerramos la conexión  ...mostrar… |
| string aux = sr[4].ToString(); | Nos devuelve la columna 4 |
| conSql.Open();  SqlDataReader sr=comando.ExecuteReader();  while (sr.Read())  {  string aux = sr["NOMBRE"].ToString();  Console.WriteLine(aux);  }  conSql.Close(); | Cambio para recorrer todos los registros |
| try  {  while (sr.Read())  {  string aux = sr["NOMBRE"].ToString();  Console.WriteLine(aux);  }  }  catch(Exception ex)  {  Console.WriteLine(ex.Message);  }  finally  {  if(conSql.State==System.Data.ConnectionState.Open){  conexionSql.Close();  }  } | Mejora para seguridad  consutar si ya esta cerrado(para que no tire error)  cierra la coneccion |
| **INSERT**  SqlConnection conexionSql = new SqlConnection("Server=. ;DataBase=MiPrimerBase;Trusted\_Connection=True");  string consulta = "INSERT INTO EMPLEADOS(NOMBRE,APELLIDO,ID\_PUESTO,SALARIO,ESTA\_ACTIVO,FECHA\_ALTA)VALUES ('Gerardo','Alanoca',3,12000,1,'2011-01-01')";  SqlCommand comando = new SqlCommand(consulta,conexionSql);  conexionSql.Open();  if (comando.ExecuteNonQuery() == 0)  {  //no se puedo insetar  }  {  //se puedo agregar  }  conexionSql.Close(); | |
| **UPDATE**  SqlConnection conSql = new SqlConnection("Server=.;DataBase=MiPrimerBase;Trusted\_Connection=True");  SqlCommand comando = new SqlCommand("UPDATE EMPLEADOS SET NOMBRE='Karen' WHERE ID\_EMPLEADO=1003", conSql);  //comando.CommandType = System.Data.CommandType.Text;  conSql.Open();  if (comando.ExecuteNonQuery() == 0)  {  //no se puedo Modificar  }  {  //se puedo Modificar  }  conSql.Close(); | |
| **DELETE**  **SqlConnection conSql = new SqlConnection("Server=.;DataBase=MiPrimerBase;Trusted\_Connection=True");**  SqlCommand comando = new SqlCommand("DELETE FROM EMPLEADOS WHERE ID\_EMPLEADO=1003", conSql);  //comando.CommandType = System.Data.CommandType.Text;  conSql.Open();  if (comando.ExecuteNonQuery() == 0)  {  //no se puedo Modificar  }  {  //se puedo Modificar  }  **conSql.Close();** | |

Evitar las inyecciones SQL

|  |
| --- |
| SqlCommand comando = new SqlCommand("UPDATE EMPLEADOS SET NOMBRE=@nombre WHERE ID\_EMPLEADO=@id", conSql);  comando.Parameters.AddWithValue("@nombre", "Esmeralda");  comando.Parameters.AddWithValue("@id", "5"); |
| Funciona para **Delete, Insert y Update.** |

FUNCIONES DAO

|  |  |
| --- | --- |
| Son clases que |  |
|  |  |

DELEGADOS(dentro del main)

Es de tipo de referencia (no de valor)

encapsulan referencias a métodos que respeten la firma del delegado.

Cuando encapsula a un método de instancia(no statico), también encapsulan al objeto que pertenece

|  |  |
| --- | --- |
| **public delegate** void MiDelegado(string varString); | |
| * Solo se declara la firma y las referencias que guardamos dentro deben respetarla * Las de negrita son obligatorio y contantes * Si lo usamos en el main va arriba del main. | |
| Dentro del main….  MiDelegado d1= new MiDelegado(MiFunc);  MiDelegado d1=MiFunc;  d1=MiFunc2; | Recordar que para darle la referencia **no va ()**  Guardamos la referencia dentro de el delegado  Lo mismo que arriba (mejor)  podemos cambiar de referencia así de fácil. |
| **Delegados Multidifusión**  MiDelegado d1= new Midelegado(MiMetodo);  MiDelegado d2= new Midelegado(MiMetodo);  MiDelegado d3= new Midelegado(MiMetodo);  MiDelegado d4 = d1 +d2;  d4+=d3; | Creamos 3 delegados  guardamos en un d4 los delegados d2 y d1  ahora añadimos d3  (no testeado) |

(dentro de una biblioteca de clases)

|  |  |
| --- | --- |
| * Declarar la firma arriaba de la clase * abajo de namespace * (prohibido olvidar)agregar la dependenci * usar el delegado como si fuera declarado dentro del main |  |

Delegados mas comunes

|  |  |
| --- | --- |
| **Action** | No devuelve nada  puede tener 0 a 16 parámetros |
| Action aux=new Action(MiMetodo);  Action aux=MiMetodo;  Action<int>aux = MiMetodo;  Action<int,string>aux = MiMetodo; | Guardamos en un delegado action una referecia  lo mismo que arriba  recibe un parámetro  recibe dos parametros |
| **Func** | Retorna valores  puede tener 0 a 16 parametros |
| Func<int> aux=new Func<int>(MiMetodo);  *“Metodo: sin parametros y retorna int”*  Func<int> aux=MiMetodo;  Func<int, int, **string**> aux=MiMetodo; | Guardamos en un delegado action una referecia  **Importante: el ultimo generico es el que nos retorna**  lo mismo que arriba  pasamos 2 int por parametros y retorna string |
| **Predicate**  Predicate<int> aux=MiMetodo; | Recive un parametro y retorna un boleano  se le indica que el unico parametro va a ser int |

|  |  |
| --- | --- |
| if(d4 is not null)//validamos  {  d4("jonathan");//llamamos  } | Los delegados pueden ser nulos y podemos solucionarlo validando si es null |

FUNCIONES LAMBDA

* funcion de una sola linea
* no se declara funciona
* van en lugar de un delegado(delegados sencillos)

|  |  |
| --- | --- |
| **Una clase**  public delegate bool CompararEnteros(int num1, int num2);  …  public int ElNumeroMayor(CompararEnteros ce, List<int> listaNumeros){….} | |
| **Program**  miClase.ElNumeroMayor(**(n1,n2)=>n1>n2**, miListaNumeros) | |
| **()=>Console.WriteLine(“.”);** | **No recibe parametros y retorna nada**  **porque WriteLine no tiene retoro** |
| **(n1,n2)=>n1>n2**  **n1=>n1\*2**  **n1=>n1.Edad>18 && n1.Edad<50**  **n1=>{**  **...logica mas elaborada…**  **}** | Lleva toda la logica de una función en poco espacio |
| **En las mas elaboradas también pueden llevar return** |  |

Hilos (subprocesos)

|  |  |  |
| --- | --- | --- |
| Task hilo1= new Task(varAction);  Task hilo1= Task.Run(varAction);  hilo1.Start();  hilo1.Wait(); | Instanciar un hilo. Estos aceptan Actions  **(atajo)** instanciar y correr el hilo  inicia el hilo  bloque la tarea hasta que finalice el hilo | |
| Action act1 = new Action(() => {Thread.Sleep(3000); Console.WriteLine("mensaje");});  Action act2 = new Action(() => { Thread.Sleep(4000); Console.WriteLine("mensaje2"); });  Task tas1 = Task.Run(act1);  Task tas2 = Task.Run(act2);  **Task.WaitAll(tas1,tas2);** | | **Como hacer wait múltiple** |
| **IMPORTANTE**  **if (lblHora.InvokeRequired)**  {  Action actAux = new Action(() => this.lblHora.Text = DateTime.Now.ToString("hh:mm:ss"));  lblHora.BeginInvoke(actAux);  }  else  {  lblHora.Text = DateTime.Now.ToString("hh:mm:ss");  **}** | | Windows Form ya corre sobre un hilo.  Al crear un hilo(task) dentro de WF este seria un hilo secundario. Y WF esperaría a que termine este y luego continua  ← solución  si requiere ser invocado desde el main thread  (osea si no esta en el hilo “principal”)  vuelve a invocar la accion desde el main thread  aca por si estamos en el “principal” |
| **CANCELATION TOKEN FRENAR LOS HILOS (WINDOWS FORM)**  **…..En los atributos**  private CancellationTokenSource tokenSource;  private CancellationToken token;  **….En el contructor**  this.tokenSource = new CancellationTokenSource();  this.token=tokenSource.Token;  **...En el task**  Task t1 = new Task(action1,**this.token**);  **...en la ejecucion**  while (!this.tokenSource.IsCancellationRequested)  **… como poner en false(en boton)**  this.tokenSource.Cancel(); | | Ademas de al action, le pasamos el token  condicional (seria como un true)  seria como un false |

MÉTODOS DE EXTENSIÓN

|  |  |
| --- | --- |
| public static class MiClaseCualquiera  {  public static string NombreMetodo(this int param)  {  return param.ToString();  }  }  -------------------------------------Uso-------------------------------------  **int** num = 21;  Console.WriteLine(num.NombreMetodo()); | Si o si debe estar en una clase estática no genérica  El metodo debe ser estatico, tipo de retorno y  this int param : hace referencia la misma variable desde cual se invoca(\*)  El retorno.  Variable cualquiera  uso |
| Agregándole mas parámetros  public static string NombreMetodo(this int param,int num2)  -------------------------------------Uso------------------------------------  Console.WriteLine(**num.NombreMetodo(2)**); | Cambios:  el this es obligatorio, los parametros agregados van despues del this.  Uso.. |