**Lesson01 what is the Type Reflection**

**Notes:-**

**(With Reflection you can handle everything about the dynamic running time creation element and reaching properties and so on)**

**1-With Reflection, we can dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties**

**Examples:-**

**1-When you drag and drop a button on a win forms or an asp.net application. The properties window uses reflection to show all the properties of the Button class. So, reflection is extensively used by IDE or UI designers.**

**2-Late binding can be achieved by using reflection. You can use reflection to dynamically create an instance of a type, about which we don't have any information at compile time. So, reflection enables you to use code that is not available at compile time.**

**3-Consider an example where we have two alternate implementations of an interface. You want to allow the user to pick one or the other using a config file. With reflection, you can simply read the name of the class whose implementation you want to use from the config file and instantiate an instance of that class. This is another example for late binding using reflection.**

**So, in short reflection can be used for type discovery (examples: - finding methods, properties, events, fields, constructors etc…) and late binding.**

**Example:-**

**Consider the Customer class example. This class has got**

**1-Two constructors**

**2-Two auto implemented properties**

**3-Two methods**

**Example:-**

**We want to show all the properties of the classes**

**public class Customer{**

**public int Id { get; set; }**

**public string Name { get; set; }**

**public Customer(int ID, string Name){**

**this.Id = ID;**

**this.Name = Name;}**

**public Customer(){**

**this.Id = -1;**

**this.Name = string.Empty;}**

**public void PrintID(){Console.WriteLine("ID = {0}", this.Id);}**

**public void PrintName(){Console.WriteLine("Name = {0}", this.Name);}}**

**//In the below code we get the property of the class dynamically at running time**

**class Program{**

**static void Main(string[] args){**

**// Get the Type Using GetType() static method**

**Type T = Type.GetType("TestingPro.Customer");**

**// Print the Type details**

**Console.WriteLine("Full Name = {0}", T.FullName);**

**Console.WriteLine("Just the Class Name = {0}", T.Name);**

**Console.WriteLine("Just the Namespace = {0}", T.Namespace);**

**Console.WriteLine();**

**// Print the list of Methods**

**Console.WriteLine("Methods in Customer Class");**

**MethodInfo[] methods = T.GetMethods();**

**foreach (MethodInfo method in methods){**

**// Print the Return type and the name of the method**

**Console.WriteLine(method.ReturnType.Name + " " + method.Name);}**

**Console.WriteLine();**

**// Print the Properties**

**Console.WriteLine("Properties in Customer Class");**

**PropertyInfo[] properties = T.GetProperties();**

**foreach (PropertyInfo property in properties){**

**// Print the property type and the name of the property**

**Console.WriteLine(property.PropertyType.Name + " " + property.Name);}**

**Console.WriteLine();**

**// Print the Constructors**

**Console.WriteLine("Constructors in Customer Class");**

**ConstructorInfo[] constructors = T.GetConstructors();**

**foreach (ConstructorInfo constructor in constructors){**

**Console.WriteLine(constructor.ToString());}**

**Console.ReadLine();}}**

**We can access to the methods and properties of the class by using the following expression MethodInfo[] and PropertyInfo[]**

**Customer cust = new Customer();**

**//The below command will store all the components of the class inside this variable**

**Type t = cust.GetType();**

**foreach (MethodInfo method in t.GetMethods()){**

**// Print the Return type and the name of the method**

**Console.WriteLine(method.ReturnType.Name + " " + method.Name);}**

**Console.WriteLine();**

**Console.WriteLine("Properties in Customer Class");**

**foreach (PropertyInfo property in t.GetProperties()){**

**// Print the property type and the name of the property**

**Console.WriteLine(property.PropertyType.Name + " " + property.Name);}**

**We can see that the methods called GetProperities() and GetMethods() and GetConstructors() using the following property type MethodInfo[] and PropertyInfo[] and ConstructorInfo[]**