C# - REFLECTION

Reflection objects are used for obtaining type information at runtime. The classes that give access to the metadata of a running program are in the **System.Reflection** namespace.

The **System.Reflection** namespace contains classes that allow you to obtain information about the application and to dynamically add types, values, and objects to the application.

Applications of Reflection

Reflection has the following applications –

- It allows view attribute information at runtime.
- It allows examining various types in an assembly and instantiate these types.
- It allows late binding to methods and properties
- It allows creating new types at runtime and then performs some tasks using those types.

Viewing Metadata

We have mentioned in the preceding chapter that using reflection you can view the attribute information.

The **MemberInfo** object of the **System.Reflection** class needs to be initialized for discovering the attributes associated with a class. To do this, you define an object of the target class, as –

```
System.Reflection.MemberInfo info = typeof(MyClass);
```

The following program demonstrates this –

```
using System;

[AttributeUsage(AttributeTargets.All)]
public class HelpAttribute : System.Attribute {
   public readonly string Url;

   public string Topic   // Topic is a named parameter {
      get {
         return topic;
      }
      set {
         topic = value;
      }
   }
   public HelpAttribute(string url)   // url is a positional parameter {
      this.Url = url;
   }
   private string topic;
}
```

```
[HelpAttribute("Information on the class MyClass")]
class MyClass {

}

namespace AttributeAppl {
   class Program {
      static void Main(string[] args) {
         System.Reflection.MemberInfo info = typeof(MyClass);
         object[] attributes = info.GetCustomAttributes(true);

         for (int i = 0; i < attributes.Length; i++) {
            System.Console.WriteLine(attributes[i]);
         }
         Console.ReadKey();
      }
   }
}
```

When it is compiled and run, it displays the name of the custom attributes attached to the class *MyClass* –

```
HelpAttribute
```

Example

In this example, we use the *DeBugInfo* attribute created in the previous chapter and use reflection to read metadata in the *Rectangle* class.

Live Demo

```
using System;
using System.Reflection;

namespace BugFixApplication {
   //a custom attribute BugFix to be assigned to a class and its members
   [AttributeUsage(
      AttributeTargets.Class |
      AttributeTargets.Constructor |
      AttributeTargets.Field |
      AttributeTargets.Method |
      AttributeTargets.Property,
      AllowMultiple = true)]

   public class DeBugInfo : System.Attribute {
      private int bugNo;
      private string developer;
      private string lastReview;
      public string message;

      public DeBugInfo(int bg, string dev, string d) {
         this.bugNo = bg;
         this.developer = dev;
         this.lastReview = d;
      }
      public int BugNo {
```

```
      get {
         return bugNo;
      }
   }
   public string Developer {
      get {
         return developer;
      }
   }
   public string LastReview {
      get {
         return lastReview;
      }
   }
   public string Message {
      get {
         return message;
      }
      set {
         message = value;
      }
   }
}
[DeBugInfo(45, "Zara Ali", "12/8/2012", Message = "Return type mismatch")]
[DeBugInfo(49, "Nuha Ali", "10/10/2012", Message = "Unused variable")]

class Rectangle {
   //member variables
   protected double length;
   protected double width;

   public Rectangle(double l, double w) {
      length = l;
      width = w;
   }
   [DeBugInfo(55, "Zara Ali", "19/10/2012", Message = "Return type mismatch")]
   public double GetArea() {
      return length * width;
   }
   [DeBugInfo(56, "Zara Ali", "19/10/2012")]
   public void Display() {
      Console.WriteLine("Length: {0}", length);
      Console.WriteLine("Width: {0}", width);
      Console.WriteLine("Area: {0}", GetArea());
   }
}//end class Rectangle

class ExecuteRectangle {
   static void Main(string[] args) {
      Rectangle r = new Rectangle(4.5, 7.5);
      r.Display();
      Type type = typeof(Rectangle);

      //iterating through the attribtues of the Rectangle class
      foreach (Object attributes in type.GetCustomAttributes(false)) {
         DeBugInfo dbi = (DeBugInfo)attributes;
```

```
            if (null != dbi) {
               Console.WriteLine("Bug no: {0}", dbi.BugNo);
               Console.WriteLine("Developer: {0}", dbi.Developer);
               Console.WriteLine("Last Reviewed: {0}", dbi.LastReview);
               Console.WriteLine("Remarks: {0}", dbi.Message);
            }
         }
         
         //iterating through the method attribtues
         foreach (MethodInfo m in type.GetMethods()) {
            
            foreach (Attribute a in m.GetCustomAttributes(true)) {
               DeBugInfo dbi = (DeBugInfo)a;
               
               if (null != dbi) {
                  Console.WriteLine("Bug no: {0}, for Method: {1}", dbi.BugNo, m.Name);
                  Console.WriteLine("Developer: {0}", dbi.Developer);
                  Console.WriteLine("Last Reviewed: {0}", dbi.LastReview);
                  Console.WriteLine("Remarks: {0}", dbi.Message);
               }
            }
         }
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result –

```
Length: 4.5
Width: 7.5
Area: 33.75
Bug No: 49
Developer: Nuha Ali
Last Reviewed: 10/10/2012
Remarks: Unused variable
Bug No: 45
Developer: Zara Ali
Last Reviewed: 12/8/2012
Remarks: Return type mismatch
Bug No: 55, for Method: GetArea
Developer: Zara Ali
Last Reviewed: 19/10/2012
Remarks: Return type mismatch
Bug No: 56, for Method: Display
Developer: Zara Ali
Last Reviewed: 19/10/2012
Remarks:
```