**Part 55 - C# Tutorial - Late binding using reflection**

**In this session we will understand**  
1. Early binding and late binding  
2. The difference between the two approaches.

**Early Binding Example:**  
using System;  
namespace Pragim  
{  
    public class MainClass  
    {  
        private static void Main()  
        {  
            Customer C1 = new Customer();  
            string fullName = C1.GetFullName("Pragim", "Tech");  
            Console.WriteLine("Full Name = {0}", fullName);  
        }  
    }  
    public class Customer  
    {  
        public string GetFullName(string FirstName, string LastName)  
        {  
            return FirstName + " " + LastName;  
        }  
    }  
}

In this example, we have the knowledge of Customer class at compile time. So, we are able to create the instance of the Customer class using the new operator. We are also able to invoke the **GetFullName**() method using **C1**. Intellisense detects the presence of this method and the number and type of parameters that need to be passed in. If you make any mistake in the name of the method, or the number and type of parameters, those mistakes will be immediately raised as compiler errors.  
  
  
**Late Binding Example:**  
using System;  
using System.Reflection;  
namespace Pragim  
{  
    public class MainClass  
    {  
        private static void Main()  
        {  
            // Load the current executing assembly as the Customer class is present in it.  
            Assembly executingAssembly = Assembly.GetExecutingAssembly();

            // Load the Customer class for which we want to create an instance dynamically

Type customerType = executingAssembly.GetType("Pragim.Customer");  
            // Create the instance of the customer type using Activator class   
            object customerInstance = Activator.CreateInstance(customerType);  
            // Get the method information using the customerType and GetMethod()  
            MethodInfo getFullName = customerType.GetMethod("GetFullNames");  
            // Create the parameter array and populate first and last names  
            string[] methodParameters = new string[2];  
            methodParameters[0] = "Pragim"; //FirstName  
            methodParameters[1] = "Tech";     //LastName  
            // Invoke the method passing in customerInstance and parameters array  
            string fullName = (string)getFullName.Invoke(customerInstance, methodParameters);  
            Console.WriteLine("Full Name = {0}", fullName);  
        }  
    }  
    public class Customer  
    {  
        public string GetFullName(string FirstName, string LastName)  
        {  
            return FirstName + " " + LastName;  
        }  
    }  
}  
  
  
**Let's assume we don't have the knowledge of Customer class at compile time, and it will be provided only at run time. In this case we need to bind to the Customer class at runtime.**  
1. Load the assembly which contains the Customer class. In our case, the Customer class is present in the same assembly as the MainClass. So, we use **Assembly.GetExecutingAssembly()** to load the current executing assembly. On the Assembly class, there are several static methods which can be used to load an assembly at runtime dynamically.

2. Next, we load the Customer class for which we want to create an instance dynamically using **executingAssembly.GetType("Pragim.Customer")**. Make sure you pass in the fully qualified name to the GetType() method, including the namespace. Otherwise you risk getting a NullReferenceException at runtime.  
3. Create the instance of the Customer class using **Activator.CreateInstance(customerType)**.  
4. Once we have the Customer instance, now get the method information which we want to invoke dynamically. we use **customerType.GetMethod("GetFullName").**  
5. The GetFullName() method expects 2 string parameters. So, we need to create a string array,  and populate it with the first and last name parameters.  
6. Finally, invoke the method passing in customerInstance and parameters array.  
  
  
If you mis-spell the method name or if you pass in the wrong number or type of parameters, you wouldn't get a compiler error, but the application crashes at runtime.  
  
  
**Difference between early and late binding:**  
1. Early binding can flag errors at compile time. With late binding there is a risk of run time exceptions.  
2. Early binding is much better for performance and should always be preferred over late binding. Use late binding only when working with onjects that are not available at compile time.