Diving into Visual Studio 2015 (Day #6): Debugging Improvements in Visual Studio 2015 (Part 2)

-Akhil Mittal

Introduction

This article is the continuation part of Debugging Improvements that were explained in Day #5 of the series. In the earlier part of the series covered topics like breakpoint configuration improvements and new improved error list in Visual Studio 2015.This article will cover another debugging improvement of Visual Studio 2015 i.e. tool window support for LINQ and Lambda expressions.

Prerequisites

Visual Studio 2015 Express has been used in this tutorial to explain the concepts and features. For samples and practice, a Visual Studio solution is created having a console application named VS2015ConsoleApplication.The console application contains a MyProduct class containing product as an entity specific basic operations like fetching the product, returning the list of products as shown below.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace VS2015ConsoleApplication
{
    public class MyProducts :IProducts
    {
        List<Product> _allProduct = new List<Product>();
        public MyProducts()
```

```
        {
            _allProduct.Add(new Product
{ProductCode="0001",ProductName="IPhone",ProductPrice="60000",ProductType="Phone",Product
Description="Apple IPhone" } );
            _allProduct.Add(new Product { ProductCode = "0002", ProductName = "Canvas",
ProductPrice = "20000", ProductType = "Phone", ProductDescription = "Micromax phone" });
            _allProduct.Add(new Product { ProductCode = "0003", ProductName = "IPad",
ProductPrice = "30000", ProductType = "Tab", ProductDescription = "Apple IPad" });
            _allProduct.Add(new Product { ProductCode = "0004", ProductName = "Nexus",
ProductPrice = "30000", ProductType = "Phone", ProductDescription = "Google Phone" });
            _allProduct.Add(new Product { ProductCode = "0005", ProductName = "S6",
ProductPrice = "40000", ProductType = "Phone", ProductDescription = "Samsung phone" });


        }


        /// <summary>
        /// FetchProduct having price greater that 3000
        /// </summary>
        /// <returns></returns>
        public List<Product> FetchProduct() => (from p in _allProduct where
Convert.ToInt32(p.ProductPrice) > 30000 select p).ToList();


        /// <summary>
        /// FetchProduct
        /// </summary>
        /// <param name="pCode"></param>
        /// <returns></returns>
        public Product FetchProduct(string pCode)
        {
            return _allProduct.Find(p => p.ProductCode == pCode);
        }


        /// <summary>
        /// FetchProduct with productCode and productName
        /// </summary>
        /// <param name="productCode"></param>
        /// <param name="productName"></param>
        /// <returns></returns>
        public Product FetchProduct(string productCode, string productName)
        {
```

```
            return _allProduct.Find(p => p.ProductCode == productCode &&
p.ProductName==productName);
        }

        public List<Product> GetProductList()
        {
            return _allProduct;
        }
    }
}
```

where IProducts is a simple interface.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace VS2015ConsoleApplication
{
    interface IProducts
    {
        Product FetchProduct(string productCode);
        Product FetchProduct(string productCode,string productName);
        List<Product> GetProductList();
    }
}
```

In the following Program.cs file the FetchProduct() method is called to get the list of all the products.

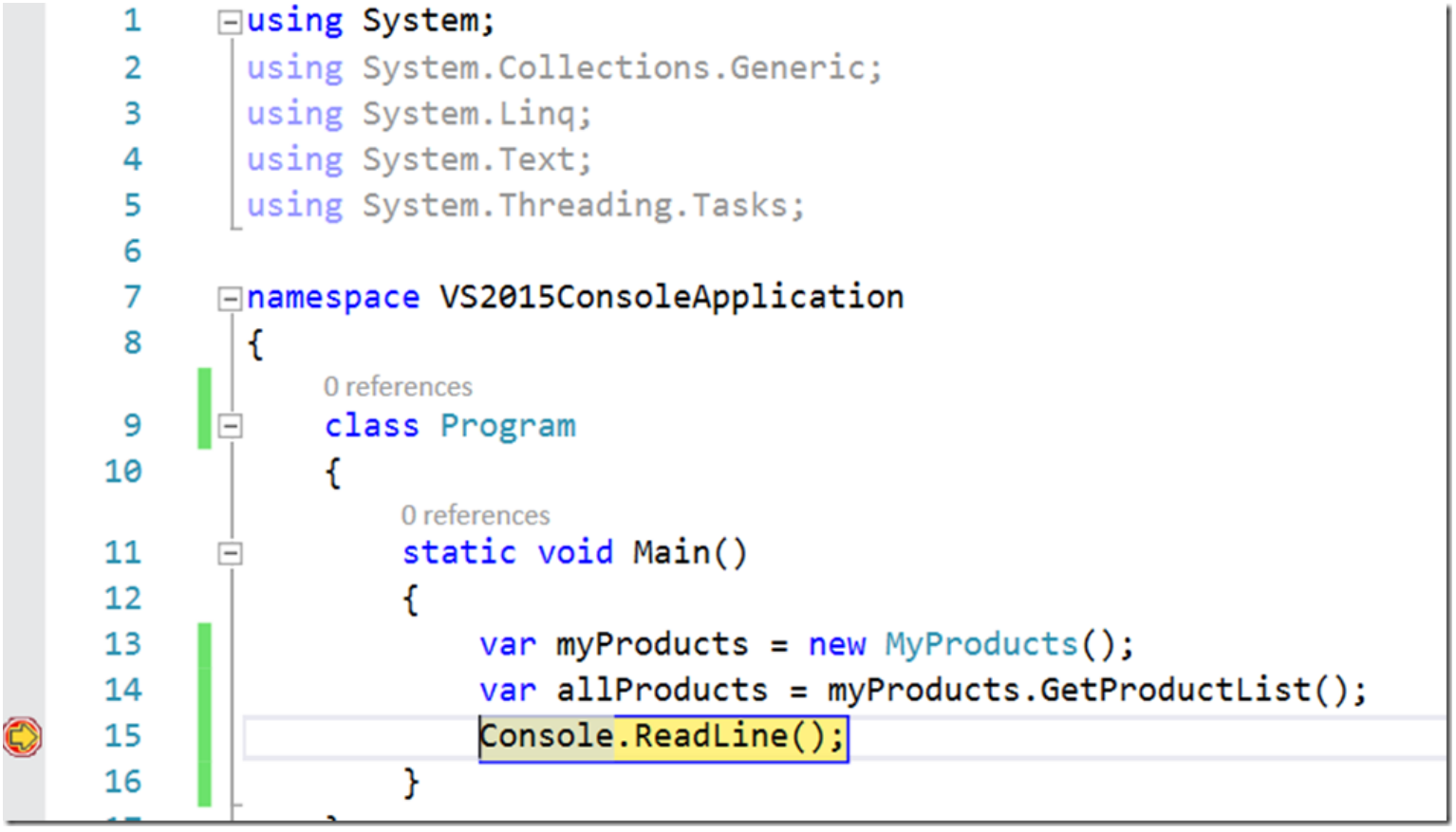
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace VS2015ConsoleApplication
{
    class Program
    {
        static void Main()
```

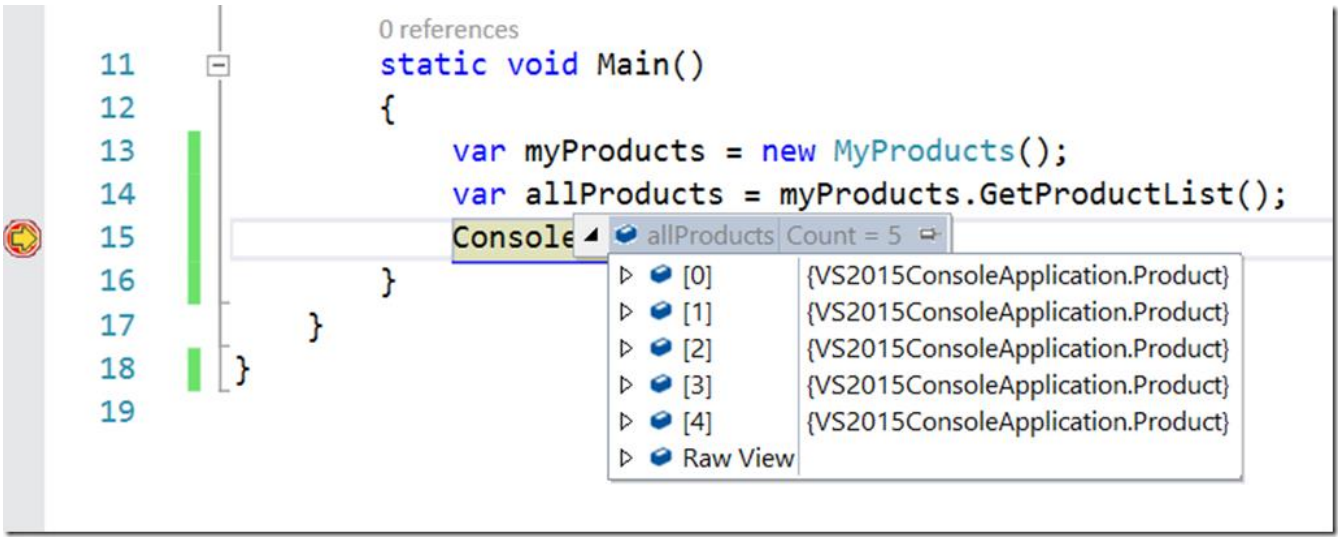
```
        {
            var myProducts = new MyProducts();
            var allProducts = myProducts.GetProductList();
            Console.ReadLine();
        }
    }
}
```

Tool window support for LINQ and lambda expressions

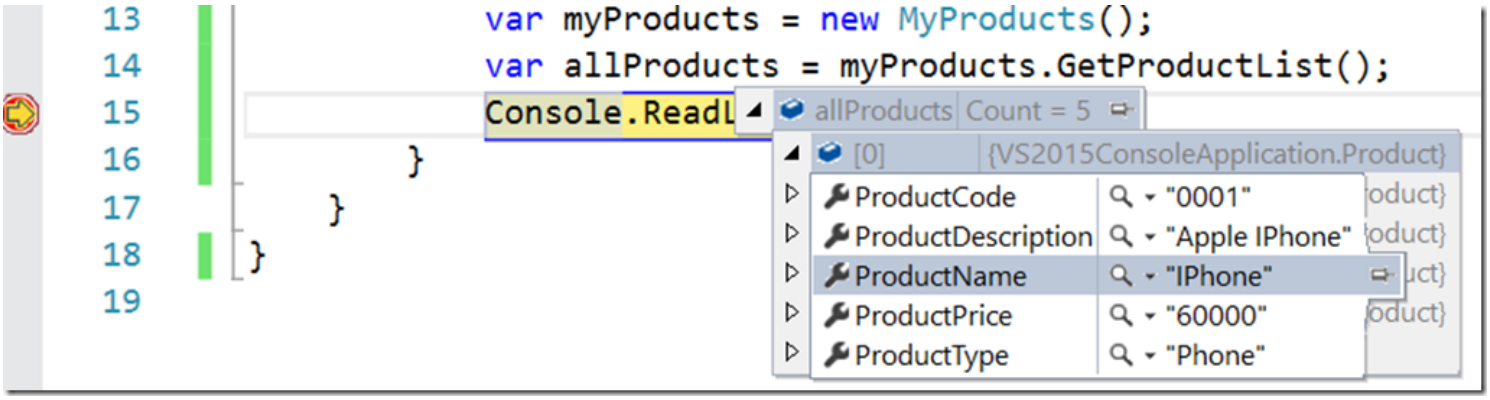
This is one of the best improvements that Visual Studio 2015 has come up with. In earlier versions of Visual Studio when a developer tries to execute/write LINQ or Lambda expressions in immediate window or watch window, the code was not supported and a message used to appear that LINQ and lambda expressions are not allowed in immediate or watch window. With the newly released product i.e. Visual Studio 2015, this limitation has been taken care of. Now a developer can take liberty to execute LINQ and Lambda expressions in immediate window. This feature proves to be very helpful in run time debugging the code, one can write LINQ queries in immediate windows at run time to select or filter the lists or objects. Let's cover the topic through practical examples. We have already a solution with a console application that fetches a list of Products. To practically check this tool support feature, place a breakpoint at Console.ReadLine() in program.cs i.e. when the list is fetched.



When we hover and try to see all products we get list of products as shown in below image.

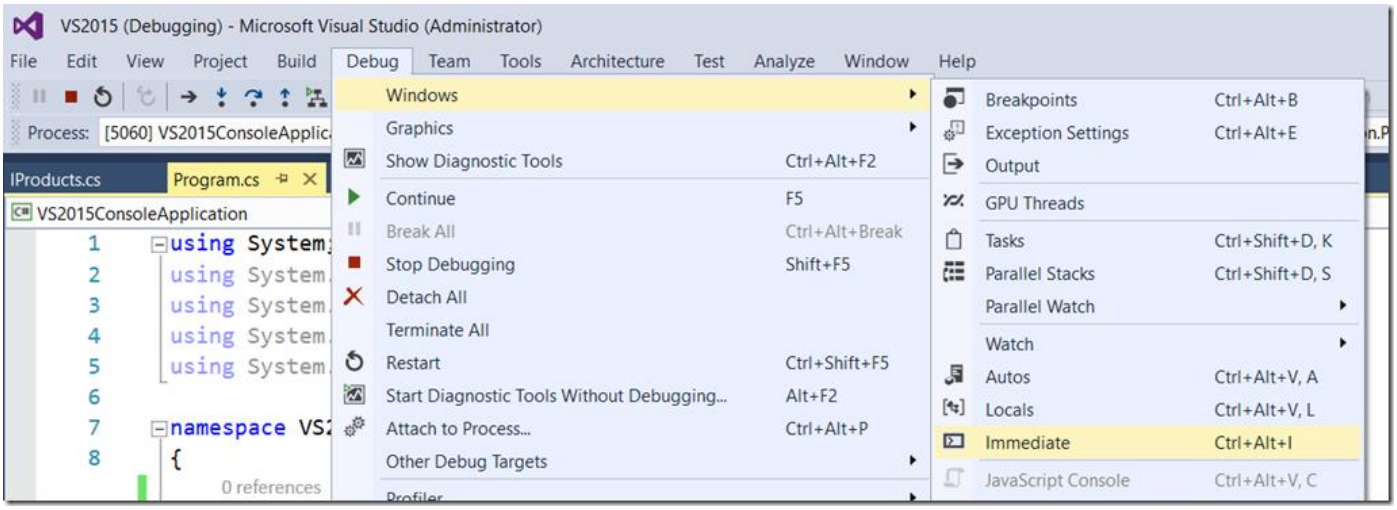


Now suppose there is a situation where developer wants to perform certain operations for debugging at this breakpoint like checking the product name, product price of each product entity or needs to see products only having price greater than 30000, then a developer has to explicitly navigate through each list item in the window opened as shown below.

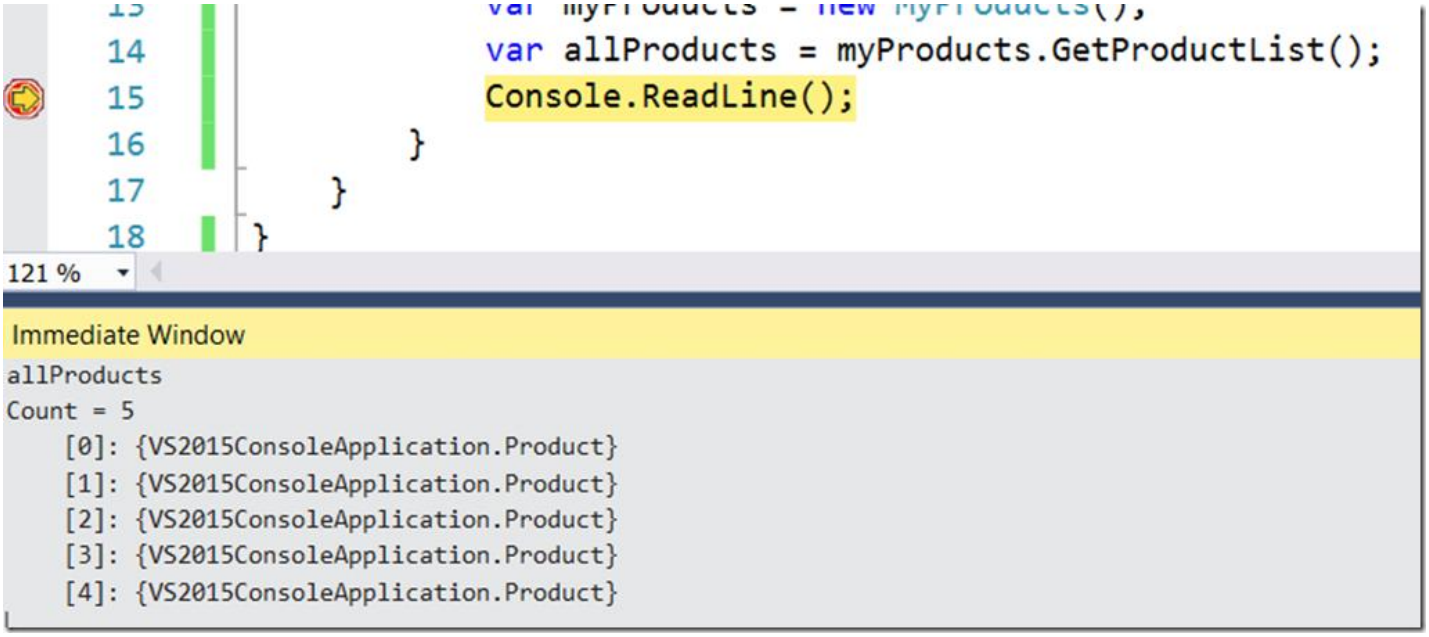


The above mentioned methodology for debugging and navigating is quite time consuming. Think about list having 1000's of records. One can leverage the support of LINQ and Lambda expressions in immediate window to expedite the debugging requirements now. Suppose one has a need to get list of all the ProductName in the list. let's see how we do this in immediate window.

Open the immediate window. You can open it through Visual Studio Debug menu as shown below. I.e. Debug->Windows->Immediate or you can use the keyboard default shortcut to open the window i.e. Ctrl+Alt+I.

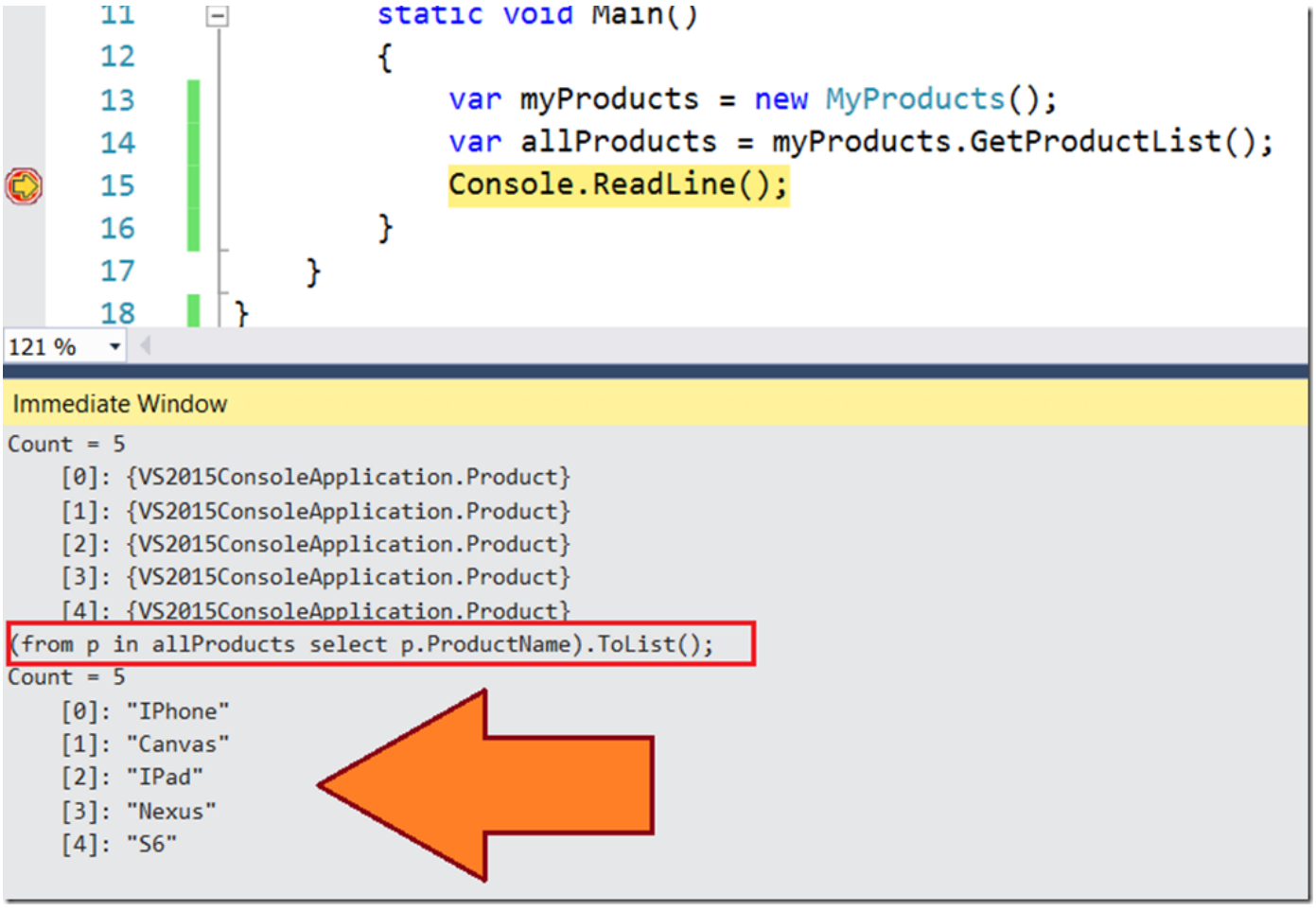


The immediate window will open at bottom of the class file and you can view the products list by typing allProducts in the window while the code execution is paused at breakpoint. You'll get the list of products as we saw while hovering the mouse over allProducts variable.



Now let's write a LINQ query to get all product names from the list of products and press enter. Notice that a list of ProductName is fetched through this LINQ query immediately.

```
(from p in allProducts select p.ProductName).ToList();
```



Likewise you can also perform filtering over the list or execute any LINQ query that you need to execute for the sake of debugging.

In earlier versions LINQ was not supported in immediate window. Now let's try this with lambda expression.

allProducts.Select(p=>p.ProductName).ToList();

We get the same result.

Conclusion

In this part of the Visual Studio 2015 series, we covered immediate window and Watch window support for LINQ and lambda expressions in Visual Studio. This is an extremely useful feature for debugging the collections or objects. We'll cover more debugging improvement features in upcoming parts. You can also use LINQ and lambda in the watch window.

For more technical articles you can reach out to my personal blog, **CodeTeddy**.