Visual Studio 2010

**Using Variance in Interfaces for Generic Collections (C# and Visual Basic)**

A covariant interface allows its methods to return more derived types than those specified in the interface. A contravariant interface allows its methods to accept parameters of less derived types than those specified in the interface.

In .NET Framework 4, several existing interfaces become covariant and contravariant. These include [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) and [IComparable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/4d7sx9hd.aspx). This enables you to reuse methods that operate with generic collections of base types for collections of derived types.

For a list of variant interfaces in the .NET Framework, see [Variance in Generic Interfaces (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dd233059.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifConverting Generic Collections

The following example illustrates the benefits of covariance support in the [IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) interface. The PrintFullName method accepts a collection of the IEnumerable<Person> type (IEnumerable(Of Person) in Visual Basic) as a parameter. However, you can reuse it for a collection of the IEnumerable<Employee> type (IEnumerable(Of Person) in Visual Basic) because Employee inherits Person.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl25_ctl00_ctl01_code');" \o "Copy Code)

' Simple hierarchy of classes.

Public Class Person

Public Property FirstName As String

Public Property LastName As String

End Class

Public Class Employee

Inherits Person

End Class

' The method has a parameter of the IEnumerable(Of Person) type.

Public Sub PrintFullName(ByVal persons As IEnumerable(Of Person))

For Each person As Person In persons

Console.WriteLine(

"Name: " & person.FirstName & " " & person.LastName)

Next

End Sub

Sub Main()

Dim employees As IEnumerable(Of Employee) = New List(Of Employee)

' You can pass IEnumerable(Of Employee),

' although the method expects IEnumerable(Of Person).

PrintFullName(employees)

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl25_ctl00_ctl02_code');" \o "Copy Code)

// Simple hierarchy of classes.

public class Person

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

public class Employee : Person { }

class Program

{

// The method has a parameter of the IEnumerable<Person> type.

public static void PrintFullName(IEnumerable<Person> persons)

{

foreach (Person person in persons)

{

Console.WriteLine("Name: {0} {1}",

person.FirstName, person.LastName);

}

}

public static void Test()

{

IEnumerable<Employee> employees = new List<Employee>();

// You can pass IEnumerable<Employee>,

// although the method expects IEnumerable<Person>.

PrintFullName(employees);

}

}

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifComparing Generic Collections

The following example illustrates the benefits of contravariance support in the [IComparer<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/8ehhxeaf.aspx) interface. The PersonComparer class implements the IComparer<Person> interface (IComparer(Of Person) in Visual Basic). However, you can reuse this class to compare a sequence of objects of the Employee type because Employee inherits Person.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl26_ctl00_ctl01_code');" \o "Copy Code)

' Simple hierarhcy of classes.

Public Class Person

Public Property FirstName As String

Public Property LastName As String

End Class

Public Class Employee

Inherits Person

End Class

' The custom comparer for the Person type

' with standard implementations of Equals()

' and GetHashCode() methods.

Class PersonComparer

Implements IEqualityComparer(Of Person)

Public Function Equals1(

ByVal x As Person,

ByVal y As Person) As Boolean \_

Implements IEqualityComparer(Of Person).Equals

If x Is y Then Return True

If x Is Nothing OrElse y Is Nothing Then Return False

Return (x.FirstName = y.FirstName) AndAlso

(x.LastName = y.LastName)

End Function

Public Function GetHashCode1(

ByVal person As Person) As Integer \_

Implements IEqualityComparer(Of Person).GetHashCode

If person Is Nothing Then Return 0

Dim hashFirstName =

If(person.FirstName Is Nothing,

0, person.FirstName.GetHashCode())

Dim hashLastName = person.LastName.GetHashCode()

Return hashFirstName Xor hashLastName

End Function

End Class

Sub Main()

Dim employees = New List(Of Employee) From {

New Employee With {.FirstName = "Michael", .LastName = "Alexander"},

New Employee With {.FirstName = "Jeff", .LastName = "Price"}

}

' You can pass PersonComparer,

' which implements IEqualityComparer(Of Person),

' although the method expects IEqualityComparer(Of Employee)

Dim noduplicates As IEnumerable(Of Employee) = employees.Distinct(New PersonComparer())

For Each employee In noduplicates

Console.WriteLine(employee.FirstName & " " & employee.LastName)

Next

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl26_ctl00_ctl02_code');" \o "Copy Code)

// Simple hierarchy of classes.

public class Person

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

public class Employee : Person { }

// The custom comparer for the Person type

// with standard implementations of Equals()

// and GetHashCode() methods.

class PersonComparer : IEqualityComparer<Person>

{

public bool Equals(Person x, Person y)

{

if (Object.ReferenceEquals(x, y)) return true;

if (Object.ReferenceEquals(x, null) ||

Object.ReferenceEquals(y, null))

return false;

return x.FirstName == y.FirstName && x.LastName == y.LastName;

}

public int GetHashCode(Person person)

{

if (Object.ReferenceEquals(person, null)) return 0;

int hashFirstName = person.FirstName == null

? 0 : person.FirstName.GetHashCode();

int hashLastName = person.LastName.GetHashCode();

return hashFirstName ^ hashLastName;

}

}

class Program

{

public static void Test()

{

List<Employee> employees = new List<Employee> {

new Employee() {FirstName = "Michael", LastName = "Alexander"},

new Employee() {FirstName = "Jeff", LastName = "Price"}

};

// You can pass PersonComparer,

// which implements IEqualityComparer<Person>,

// although the method expects IEqualityComparer<Employee>.

IEnumerable<Employee> noduplicates =

employees.Distinct<Employee>(new PersonComparer());

foreach (var employee in noduplicates)

Console.WriteLine(employee.FirstName + " " + employee.LastName);

}

}