Programming with C#

# Using linq, Practice

Retrieve data from a BinaryTree by using the extension methods

1. Start Microsoft Visual Studio 2013 if it is not already running.
2. Navigate to the folder where you saved the companion content.
3. In the Chapter 21 folder open either the Windows 7 or Windows 8 folder.
4. Open the QueryBinaryTree project.
5. In Solution Explorer, right-click the QueryBinaryTree project, point to Add, and then click Class. In the Add New Item – Query BinaryTree dialog box, in the Name box, type Employee.cs and then click Add.
6. Add the automatic properties shown below in bold to the Employee class:

class Employee  
{   
 **public string FirstName { get; set; }   
 public string LastName { get; set; }   
 public string Department { get; set; }   
 public int Id { get; set; }**  
}

1. Add the ToString method shown in bold in the code that follows to the Employee class. Types in the .NET Framework use this method when converting the object to a string representation, such as when displaying it by using the Console.WriteLine statement.

class Employee  
{   
 ...   
 **public override string ToString()   
 {   
 return String.Format("Id: {0}, Name: {1} {2}, Dept: {3}", this.Id, this.FirstName, this.LastName, this.Department);   
 }**  
}

1. Modify the definition of the Employee class to implement the IComparable<Employee> interface, as shown here:

class Employee : IComparable<Employee>  
{  
}

This step is necessary because the BinaryTree class specifies that its elements must be “comparable.”

1. Right-click the *IComparable<Employee>* interface in the class definition, point to Implement Interface, and then click Implement Interface Explicitly.

This action generates a default implementation of the CompareTo method. Remember that the BinaryTree class calls this method when it needs to compare elements when inserting them into the tree.

1. Replace the body of the CompareTo method with the code that follows shown in bold. This implementation of the CompareTo method compares Employee objects based on the value of the Id field.

int IComparable<Employee>.CompareTo(Employee other)  
{   
 **if (other == null)   
 {   
 return 1;   
 }   
 if (this.Id > other.Id)   
 {   
 return 1;   
 }   
 if (this.Id < other.Id)   
 {   
 return -1;   
 }   
 return 0;**  
}

*Note For a description of the IComparable<T> interface, refer to Chapter 19.*

1. In Solution Explorer, right-click the QueryBinaryTree solution, point to Add, and then click Existing Project. In the Add Existing Project dialog box, move to the folder Microsoft Press\Visual CSharp Step By Step\Chapter 21\Windows X\BinaryTree in your Documents folder, click the BinaryTree project, and then click Open.

The BinaryTree project contains a copy of the enumerable BinaryTree class that you implemented in Chapter 19.

1. In Solution Explorer, right-click the QueryBinaryTree project, and then, on the shortcut menu that opens, click Add Reference. In the Reference Manager – QueryBinaryTree dialog box, in the left pane, click Solution. In the middle pane, select the BinaryTree project, and then click OK.
2. Display the Program.cs file for the QueryBinaryTree project in the Code and Text Editor window, and verify that the list of using directives at the top of the file includes the following line of code:

using System.Linq;

1. Add the following using directive that brings the BinaryTree namespace into scope to the list at the top of the Program.cs file:

using BinaryTree;

1. In the doWork method in the Program class, remove the // TODO: comment and add the following statements shown in bold type to construct and populate an instance of the BinaryTree class:

static void doWork()   
{   
 **Tree<Employee> empTree = new Tree<Employee>( new Employee { Id = 1, FirstName = "Kim", LastName = "Abercrombie", Department = "IT" });   
 empTree.Insert( new Employee { Id = 2, FirstName = "Jeff", LastName = "Hay", Department = "Marketing" });   
 empTree.Insert( new Employee { Id = 4, FirstName = "Charlie", LastName = "Herb", Department = "IT" });   
 empTree.Insert( new Employee { Id = 6, FirstName = "Chris", LastName = "Preston", Department = "Sales"});   
 empTree.Insert( new Employee { Id = 3, FirstName = "Dave", LastName = "Barnett", Department = "Sales" });   
 empTree.Insert( new Employee { Id = 5, FirstName = "Tim", LastName = "Litton", Department="Marketing" });**  
}

1. Add the following statements shown in bold to the end of the doWork method. This code invokes the Select method to list the departments found in the binary tree.

static void doWork()   
{   
 ...   
 **Console.WriteLine("List of departments");   
 var depts = empTree.Select(d => d.Department);   
 foreach (var dept in depts)   
 {   
 Console.WriteLine("Department: {0}", dept);   
 }**   
}

1. On the Debug menu, click Start Without Debugging.
2. The application should output the following list of departments:

List of departments   
Department: IT   
Department: Marketing   
Department: Sales   
Department: IT   
Department: Marketing   
Department: Sales

Each department occurs twice because there are two employees in each department. The order of the departments is determined by the CompareTo method of the Employee class, which uses the Id property of each employee to sort the data. The first department is for the employee with the Id value 1, the second department is for the employee with the Id value 2, and so on.

1. Press Enter to return to Visual Studio 2013.
2. In the doWork method in the Program class, modify the statement that creates the enumerable collection of departments as shown in bold in the following example:

var depts = empTree.Select(d => d.Department).**Distinct();**

The Distinct method removes duplicate rows from the enumerable collection.

1. On the Debug menu, click Start Without Debugging.
2. Verify that the application now displays each department only once, like this:

List of departments  
Department: IT  
Department: Marketing  
Department: Sales

1. Press Enter to return to Visual Studio 2013.
2. Add the following statements shown in bold to the end of the doWork method. This block of code uses the Where method to filter the employees and return only those in the IT depart¬ment. The Select method returns the entire row rather than projecting specific columns.

static void doWork()  
{   
 ...   
 **Console.WriteLine("\nEmployees in the IT department");   
 var ITEmployees = empTree.Where(e => String.Equals(e.Department, "IT")) .Select(emp => emp);   
 foreach (var emp in ITEmployees)   
 {   
 Console.WriteLine(emp);   
 }**  
}

1. Add the code shown below in bold to the end of the doWork method, after the code from the preceding step. This code uses the GroupBy method to group the employees found in the bi¬nary tree by department. The outer foreach statement iterates through each group, displaying the name of the department. The inner foreach statement displays the names of the employ¬ees in each department.

static void doWork()  
{   
 ...   
 **Console.WriteLine("\nAll employees grouped by department");   
 var employeesByDept = empTree.GroupBy(e => e.Department);  
 foreach (var dept in employeesByDept)**

**{**

**Console.WriteLine("Department: {0}", dept.Key);**

**foreach (var emp in dept)**

**{**

**Console.WriteLine("\t{0} {1}", emp.FirstName, emp.LastName);**

**}**

**}**

}

1. On the Debug menu, click Start Without Debugging. Verify that the output of the application looks like this:

List of departments  
Department: IT  
Department: Marketing  
Department: Sales  
  
Employees in the IT department  
Id: 1, Name: Kim Abercrombie, Dept: IT  
Id: 4, Name: Charlie Herb, Dept: IT  
  
All employees grouped by department  
Department: IT   
 Kim Abercrombie   
 Charlie Herb  
Department: Marketing   
 Jeff Hay   
 Tim Litton  
Department: Sales   
 Dave Barnett   
 Chris Preston

1. Press Enter to return to Visual Studio 2013.

Retrieve data from a BinaryTree by using query operators

1. In the doWork method, comment out the statement that generates the enumerable collection of departments, and replace it with the equivalent statement shown in bold, using the from and select query operators:

// var depts = empTree.Select(d => d.Department).Distinct();  
var depts = (from d in empTree   
 select d.Department).Distinct();  
  
// var ITEmployees =

// empTree.Where(e => String.Equals(e.Department, "IT"))

// .Select(emp => emp);

**var ITEmployees = from e in empTree**

**where String.Equals(e.Department, "IT")**

**select e;**

1. Comment out the statement that generates the enumerable collection grouping employees by department, and replace it with the following code shown in bold:  
     
   // var employeesByDept = empTree.GroupBy(e => e.Department);  
   **var employeesByDept = from e in empTree   
    group e by e.Department;**
2. On the Debug menu, click Start Without Debugging. Verify that the program displays the same results as before.

List of departments  
Department: IT  
Department: Marketing  
Department: Sales  
Employees in the IT department  
Id: 1, Name: Kim Abercrombie, Dept: IT  
Id: 4, Name: Charlie Herb, Dept: IT  
All employees grouped by department  
Department: IT   
 Kim Abercrombie   
 Charlie Herb  
Department: Marketing   
 Jeff Hay   
 Tim Litton  
Department: Sales   
 Dave Barnett   
 Chris Preston

1. Press Enter to return to Visual Studio 2013.