Иллюстрации к курсу лекций по дисциплине «Программирование на C#»

Introduction to LINQ (*Language INtegrated Query)*

Использованы материалы пособия Daniel Solis, Illustrated C#

Введение в LINQ

Пример запроса LINQ

```
public static void Main() {
    int[] numbers = { 2, 12, 5, 15 };
    // определение запроса
    IEnumerable<int> lowNums =
                              from n in numbers
                               where n < 10
                               select n;
    // Выполнение запроса
    foreach (var x in lowNums)
        Console.Write($"{ x }, ");
}
```

Результат работы:

2, 5,

Контекстно-зависимые служебные слова языка LINQ

from

where

select

group

into

orderby

join

let

in

on

equals

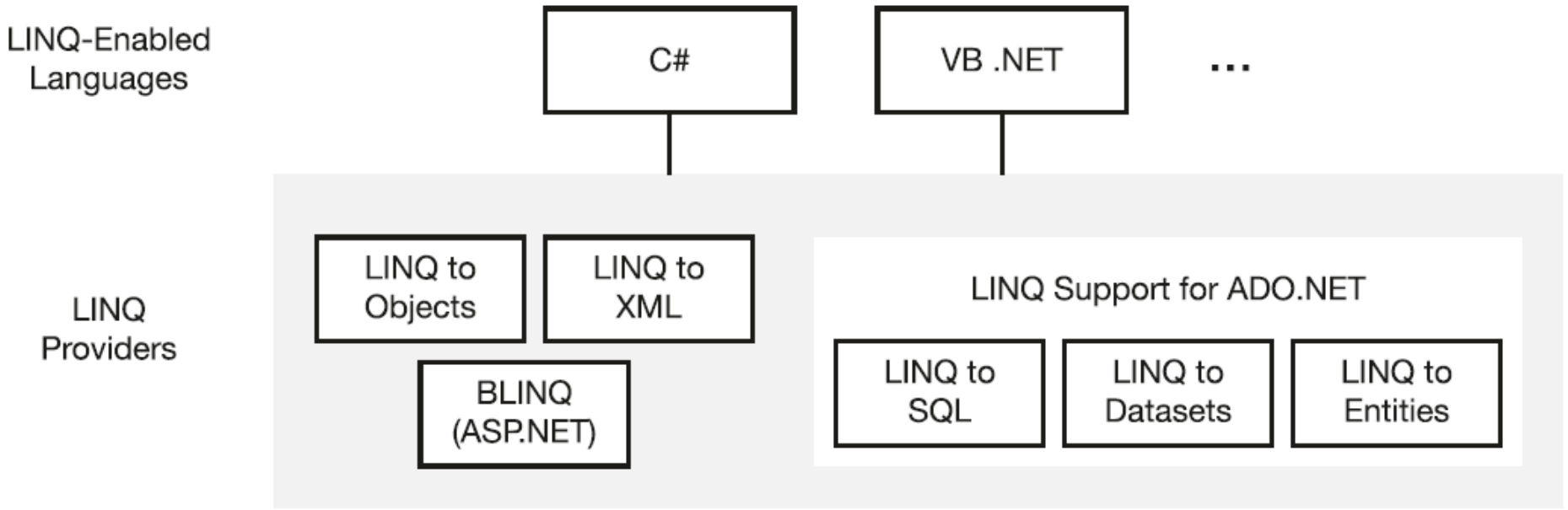
by

ascending

descending

https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/query-keywords

Провайдеры LINQ



LINQ API – это набор методов расширения из классов System.Linq.**Enumerable** и System.Linq.**Queryable**

Анонимные типы

Анонимный тип – создаваемый компилятором ссылочный тип, инкапсулирующий свойства только для чтения.

Инициализатор объекта

```
new TypeName(ArgList) { FieldOrProp = InitExpr, FieldOrProp = InitExpr, ...}
new TypeName { FieldOrProp = InitExpr, FieldOrProp = InitExpr, ...}
```

Инициализатор поля/свойства **Инициализатор поля/свойства**

Нет имени класса **Инициализатор анонимного объекта**

```
new       { FieldProp = InitExpr, FieldProp = InitExpr, ...}
```

Инициализатор **Инициализатор**

Создается объект **ссылочного** типа со свойствами только для чтения!

https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/anonymous-types

Пример анонимного типа

```
public static void Main()
{
// var обязательно!                Инициализатор объекта
    ↓                                   ↓
    var student = new { Name = "Mary Jones", Age = 19, Major = "History" };
    Console.WriteLine($"{ student.Name }, Age { student.Age }, Major: {student.Major}");
}
```

Результат работы:
Mary Jones, Age 19, Major: History

Инициализация анонимного объекта

```
public class Other {
        public static string Name = "Mary Jones";
}

public static void Main() {
        string Major = "History";

//                Присваивание  ↓     идентификатор ↓
        var student = new { Age = 19, Other.Name, Major };
//                          доступ члену класса ↑

        Console.WriteLine("{0}, Age {1}, Major: {2}",
                student.Name, student.Age, student.Major);
}
```

Об эквивалентности объявлений

```
var student = new { Age = 19, Other.Name, Major };
```

равносильно

```
var student = new { Age = 19, Name = Other.Name, Major = Major };
```

Вывод объекта анонимного типа

```
public static void Main() {
    var rectangles = new[]      // массив объектов анонимного типа
    {       new { a = 3, b = 6},
            new { a = 4, b = 1}
    };

    foreach (var rec in rectangles)
        Console.WriteLine(rec);   // rec.ToString()
}
```

Результаты выполнения программы:

{ a = 3, b = 6 }

{ a = 4, b = 1 }

Методы LINQ

Name Category	Метод	Описание категории
Restriction **Ограничение**	**Where**	Возвращает подмножество объектов из последовательности, выбирая их на основе критерия (предиката).
Projection **Отображение**	**Select** **SelectMany**	Выбирает элементы последовательности и создает из них другую последовательность, с элементами, возможно, другого типа
Partitioning **Разбиение**	**Take** **TakeWhile** **Skip** **SkipWhile**	Выбирает (возвращает) или отбрасывает (пропускает) объекты последовательности
Join **Соединение**	**Join** **GroupJoin**	Возвращает перечислимый (IEnumerable<T>) объект, который соединяет элементы двух последовательностей, согласно заданному критерию.
Concatenation **Конкатенация**	**Concat**	“Склеивает” две последовательности в одну

Методы LINQ (продолжение)

Name Category	Метод	Описание категории
Ordering **Упорядочивание**	**OrderBy** **OrderByDescending** **ThenBy** **ThenByDescending** **Reverse**	Упорядочивает элементы последовательности на основе заданного критерия.
Grouping **Группировка**	**GroupBy**	Группирует элементы последовательности на основе заданного критерия.
Set **Множества**	**Distinct** **Union** **Intersect** **Expect**	Выполняет на элементах последовательности (последовательностей) телретико-множественные операции.
Conversion **Преобразования**	**Cast** **OfType** **AsEnumerable** **ToArray** **ToList** **ToDictionary** **ToLookup**	Преобразует последовательности к различным формам, таким как массивы, списки, словари.
Equality **Эквивалентность**	**SequenceEqual**	Сравнивает (на равенство) две последовательности.

Методы LINQ (продолжение)

Name Category	Метод	Описание категории
Element **Элемент**	**DefaultIfEmpty** **First** **FirstOrDefault** **Last** **LastOrDefault** **Single** **SingleOrDefault** **ElementAt** **ElementAtOrDefault**	Возвращает конкретный элемент последовательности.
Generation **Генерация**	**Range** **Repeat** **Empty**	Генерирует последовательности.
Quantifiers **Квантификаторы**	**Any** **All** **Contains**	Возвращает логические значения, определяющие истинность заданного предиката на текущей последовательности
Aggregate **Агрегация**	**Count** **LongCount** **Sum** **Min** **Max** **Average** **Aggregate**	Возвращает отдельное значение, представляющее запрашиваемую характеристику последовательности.

Сигнатуры стандартных методов LINQ

всегда public + static

имя и обобщенный параметр

первый параметр

```
public static int Count<T>(this IEnumerable<T> source);

public static T First<T>(this IEnumerable<T> source);

public static IEnumerable<T> Where<T>(this IEnumerable<T> source, ... );
```

тип возврата

признак метода расширения

Вызовы методов LINQ, как стандартные “операции”

```
class Program
{
    static int[] numbers = new int[] { 2, 4, 6 };

    static void Main()
    {
        int total = numbers.Sum();
        int howMany = numbers.Count();
        Console.WriteLine("Total: {0}, Count: {1}", total, howMany);
    }
}
```

Результат работы программы:
Total: 12, Count: 3

Пример вызовов методов LINQ

```
using System.Linq;
static void Main()
{
    int[] intArray = new int[] { 3, 4, 5, 6, 7, 9 };
                                  ссылка на массив
                                        ↓
    var count1 = Enumerable.Count(intArray);   // непосредственный вызов
    var firstNum1 = Enumerable.First(intArray); // непосредственный вызов
    var count2 = intArray.Count(); // вызов в качестве метода расширения
    var firstNum2 = intArray.First(); // вызов в качестве метода расширения
                          ↑
                  массив в качестве "расширенного" объекта
    Console.WriteLine("Count: {0}, FirstNumber: {1}", count1, firstNum1);
    Console.WriteLine("Count: {0}, FirstNumber: {1}", count2, firstNum2);
}
```

Результат работы программы:

Count: 6, FirstNumber: 3

Count: 6, FirstNumber: 3

Экземплярная и статическая формы вызова метода

Прототип метода 1 для операции **Count**:

```
public static int Count<T>(this IEnumerable<T> source);
```

Формы применения:

```
var count1 = Linq.Enumerable.Count(intArray); // статический метод
var count2 = intArray.Count(); // экземплярная форма вызова
```

Перегрузка для метода **Count**:

```
public static int Count<T>(this IEnumerable<T> source,
                                    Func<T, bool> predicate );
```

↑

обобщеный делегат

Предопределенные типы делегатов Func

```
public delegate TR Func<out TR> ( );

public delegate TR Func<in T1, out TR > ( T1 a1 );

public delegate TR Func<in T1, in T2, out TR > ( T1 a1, T2 a2 );
```

↑ тип возврата

↑ типизирующие параметры

↑ параметры метода

```
public static int Count<T>(this IEnumerable<T> source,
                           Func<T, bool> predicate );
```

↑ тип параметра

↑ тип возврата

Предопределенные типы делегатов Action

```
public delegate void Action ( );

public delegate void Action<in T1> ( T1 a1 );

public delegate void Action<in T1, in T2> ( T1 a1, T2 a2 );

public delegate void Action<in T1, in T2, in T3>( T1 a1, T2 a2, T3 a3 );
```

Делегаты в качестве параметров (2)

public static int Count<T>(this IEnumerable<T> source, **Func<T, bool> predicate**);

↑

обобщеный делегат

```
static void Main()
{
    int[] intArray = new int[] { 3, 4, 5, 6, 7, 9 };
    var countOdd = intArray.Count(n => n % 2 == 1);
                                  ↑
        лямбда-выражение для определения нечетных значений
    Console.WriteLine("Count of odd numbers: {0}", countOdd);
}
```

Результат работы программы:
Count of odd numbers: 4

Пример с делегатом

```
class Program {
    static bool IsOdd(int x) // метод для делегата
    {   return x % 2 == 1; }  // true для нечетных

    static void Main()  {
        int[] intArray = new int[] { 3, 4, 5, 6, 7, 9 };
        Func<int, bool> myDel = new Func<int, bool>(IsOdd);
        int countOdd = intArray.Count(myDel); // исп. делегат
        Console.WriteLine("Кол-во нечетных: {0}", countOdd);
    }
}
```

Результат работы программы:

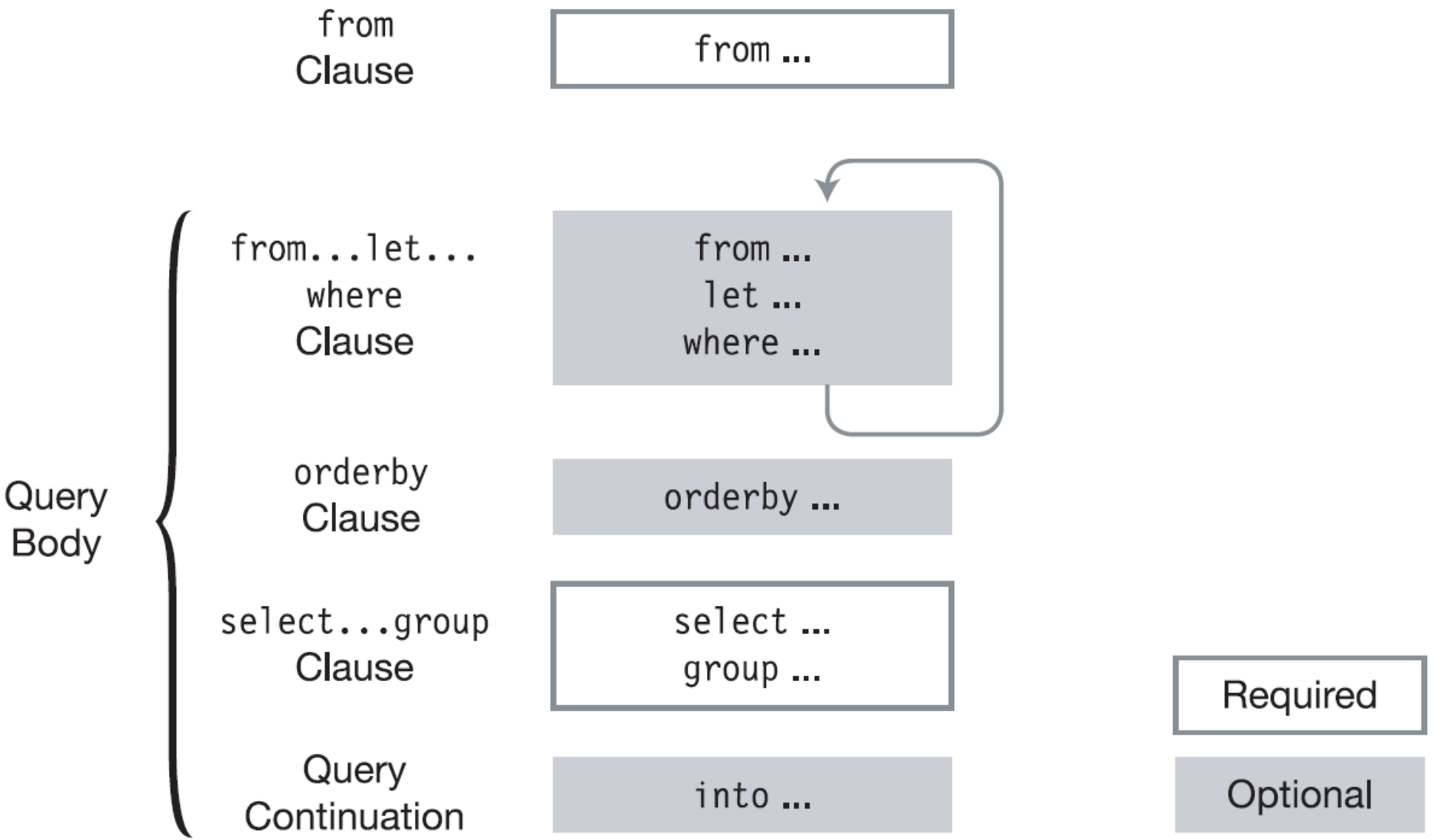
4

Смешение синтаксиса запросов и вызовов методов

```
static void Main() {
    var numbers = new int[] { 2, 6, 4, 8, 10 };

    int howMany = (from n in numbers
                   where n < 7
                   select n).Count();

    Console.WriteLine("Count: {0}", howMany);
}
```

Результат работы программы:
Count: 3

Структура выражений в запросах



Синтаксис запросов и методов

```
public static void Main() {
    int[] numbers = { 2, 5, 28, 31, 17, 16, 42 };
    // Результат - коллекция:
    var numsQuery = from n in numbers // запрос
                    where n < 20
                    select n;

    // Результат - коллекция:
    var numsMethod = numbers.Where(x => x < 20); // метод

    // Результат - число - количество выбранных значений:
    int numsCount = (from n in numbers // комбинация
                         where n < 20
                         select n).Count();
    // см. след. слайд...
}
```

Результаты запросов

```
foreach (var x in numsQuery) // 2, 5, 17, 16,
      Console.Write("{0}, ", x);
Console.WriteLine();

foreach (var x in numsMethod) // 2, 5, 17, 16,
      Console.Write("{0}, ", x);
Console.WriteLine();

Console.WriteLine($"numsCount = {numsCount}");
```

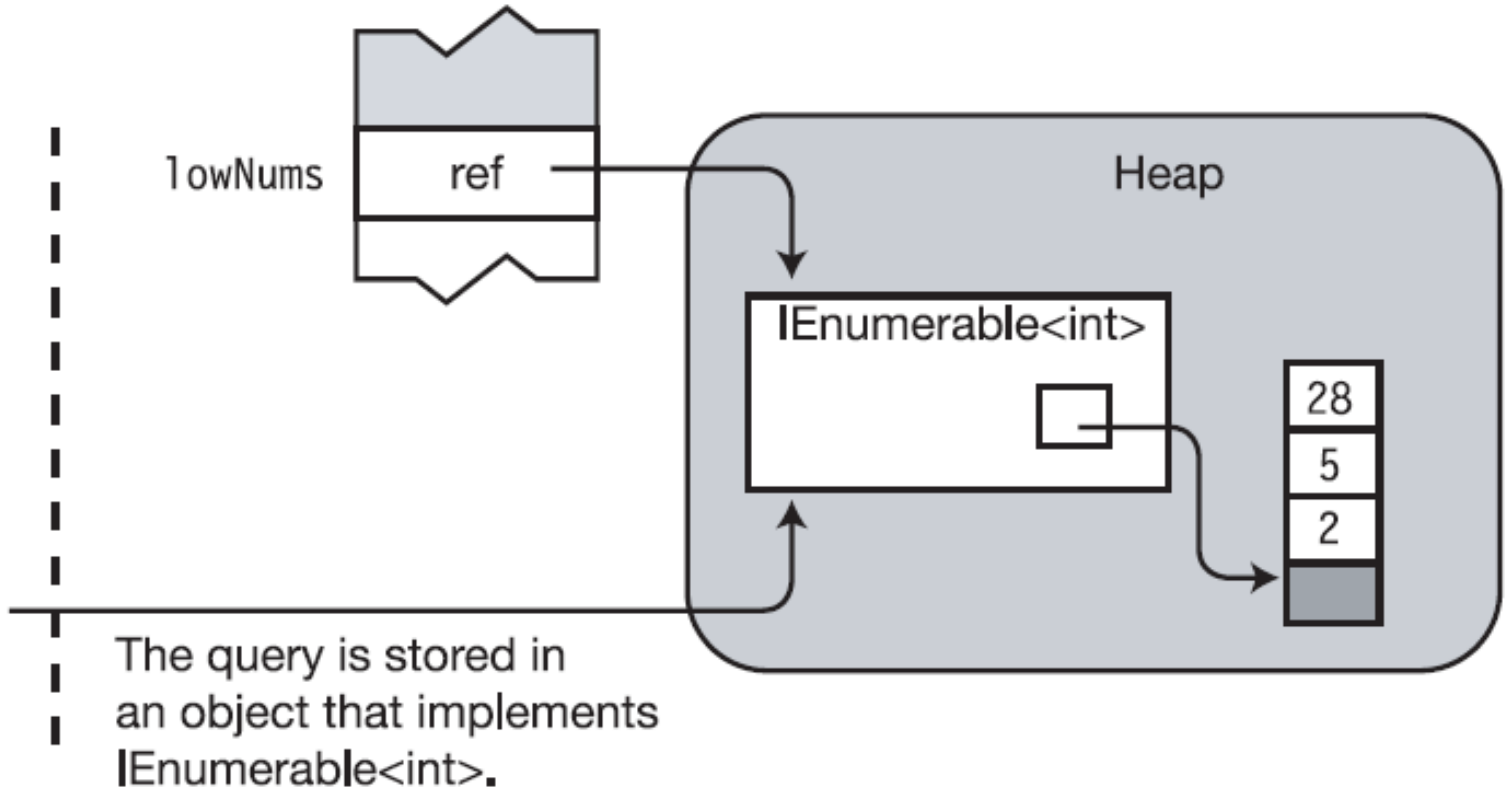
Результаты выполнения программы :
2, 5, 17, 16,
2, 5, 17, 16,
numsCount = 4

Переменные запросов

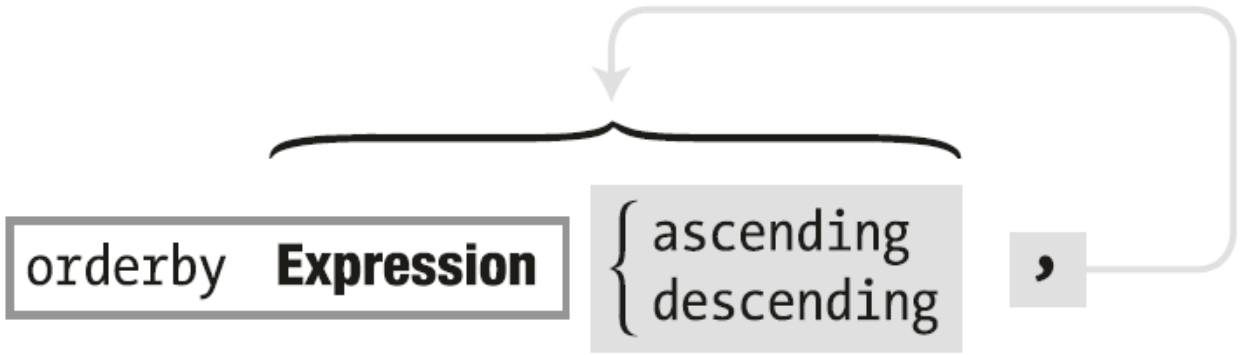
```
public static void Main ()
{
    int[] numbers = { 2, 5, 28 };


    // Возвращает ссылку с типом перечисления:
    IEnumerable<int> lowNums = from n in numbers
                               where n < 20
                               select n;


    int numsCount = (from n in numbers // Возвращает int
                     where n < 20
                     select n).Count();

    Console.WriteLine($"numsCount = {numsCount}"); // Вывод: 2
}
```

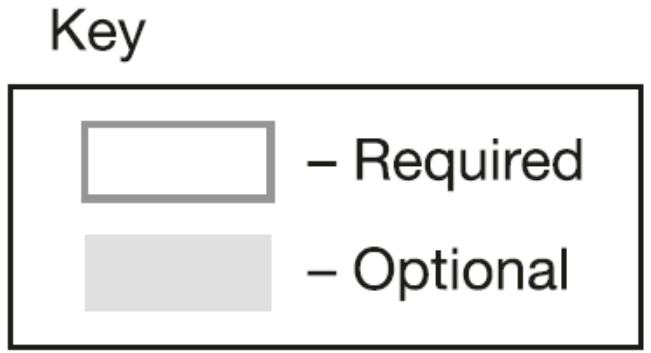
Переменная запроса

```
static void Main( )
{
   int[] numbers = { 2, 5, 28 };

   IEnumerable<int> lowNums =
                  from n in numbers
                  where n < 20
                  select n;

   ...
}
```



Предложение orderby





Сортировка с orderby

```
static void Main()
{
    var students = new[] { // массив объектов анонимного типа
        new { LName="Jones", FName="Mary", Age=19, Major="History" },
        new { LName="Smith", FName="Bob", Age=20, Major="CompSci" },
        new { LName="Fleming", FName="Carol", Age=21, Major="History" }
    };

    var query = from student in students
                orderby student.Age, student.FName descending    // сортировка
                select student;
    foreach (var s in query)
    {
        Console.WriteLine("{0}, {1}: {2} - {3}",
                    s.LName, s.FName, s.Age, s.Major);
    }
} // вывод: Jones, Smith, Fleming …
```

Анонимные типы в запросах

```
select new { s.LastName, s.FirstName, s.Major };
```

↑

анонимный тип

```
var query = from s in students
            select new { s.LName, s.FName, s.Major };
foreach (var q in query)
    Console.WriteLine("{0} {1} -- {2}", q.FName, q.LName, q.Major);
```

Результат работы программы:

Mary Jones -- History
Bob Smith -- CompSci
Carol Fleming -- History

Завершение запроса: select или group

```
select  Expression
```

```
group  Expression1  by  Expression2
```

```
var query = from s in students
            select s.LName;
foreach (var q in query)
    Console.WriteLine(q);
```

Результат работы:

Jones
Smith
Fleming

Использование group

group student **by** student.Major;

↑ ↑

кл. слово кл. слово

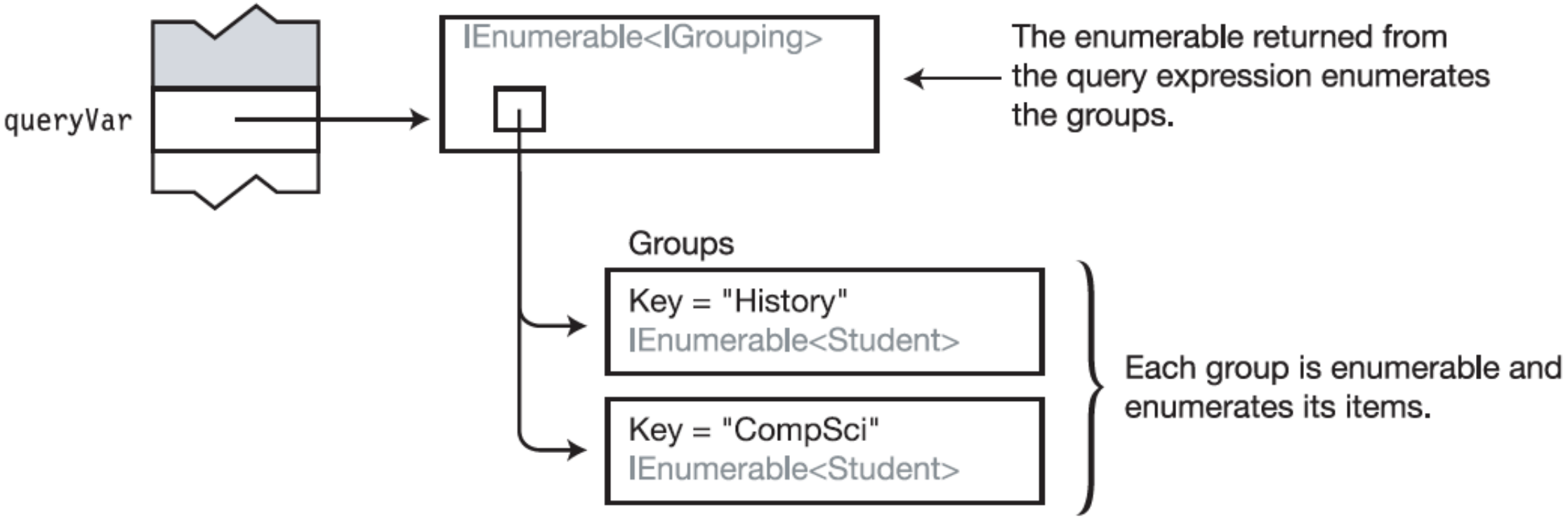
****

```
var queryVar =
    from student in students
    group student by student.Major;
foreach (var s in queryVar) { // перечисляем группы
    Console.WriteLine($"Ключ группы {s.Key}");     // s.Key – ключ группы
    foreach (var t in s)     // перечислим элементы группы
        Console.WriteLine($"\t {t.LName}, {t.FName}");
}
```

Результат выполнения:

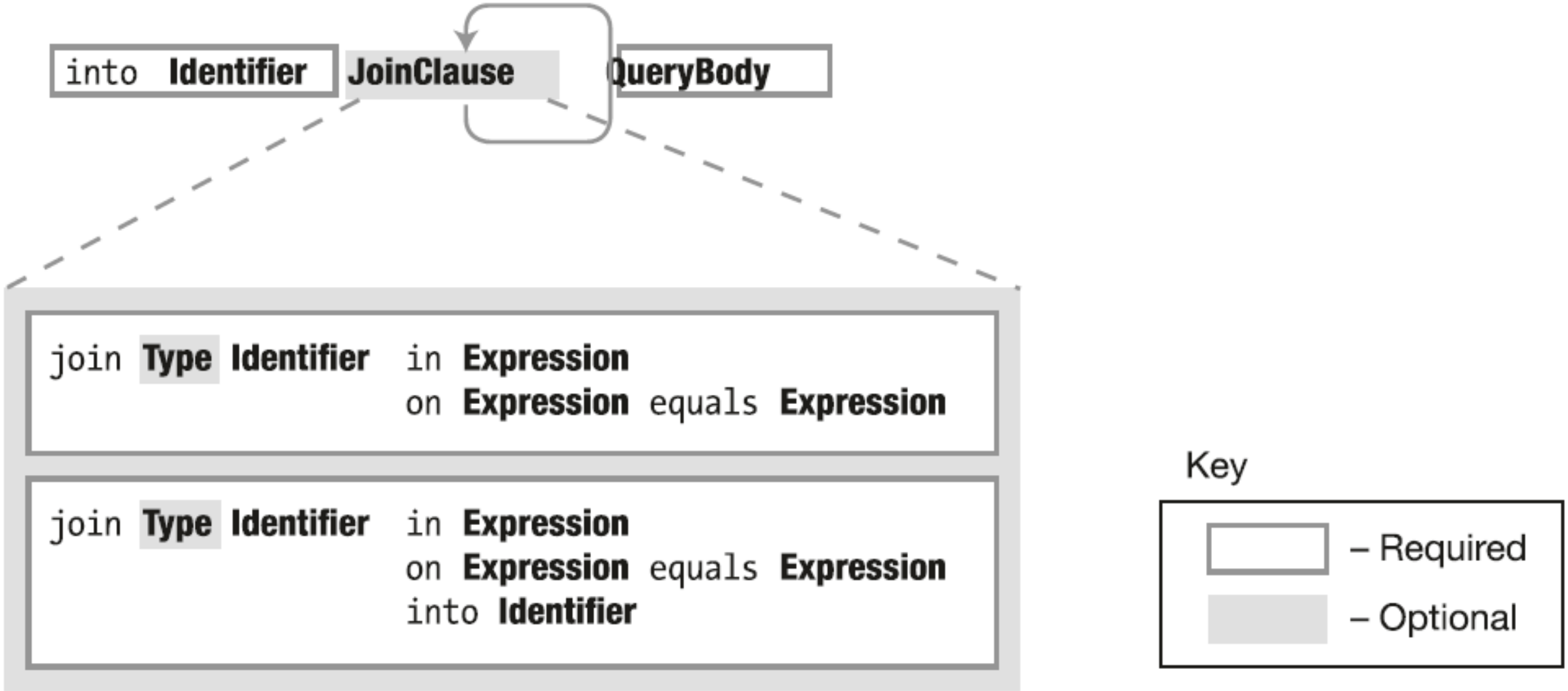
History
    Jones, Mary
    Fleming, Carol
CompSci
    Smith, Bob

Схема работы group




```
public interface IGrouping<out TKey,out TElement> :
        System.Collections.Generic.IEnumerable<out TElement>
{
        public TKey Key { get; }
}
```

Продолжение запроса (схема)



Продолжение запроса

```
public static void Main ()
{
    var groupA = new[] { 3, 4, 4, 5, 6 };
    var groupB = new[] { 4, 4, 5, 6, 7 };
    var someInts = from a in groupA
                   join b in groupB on a equals b
                   into groupAandB  // продолжение запроса
                   from c in groupAandB
                   select c;
    foreach (var a in someInts)
        Console.Write("{0} ", a);
    Console.WriteLine();
}
```

Результат вывода:

4 4 4 4 5 6

Предложение from (1)

объявление переменной итерации

↓

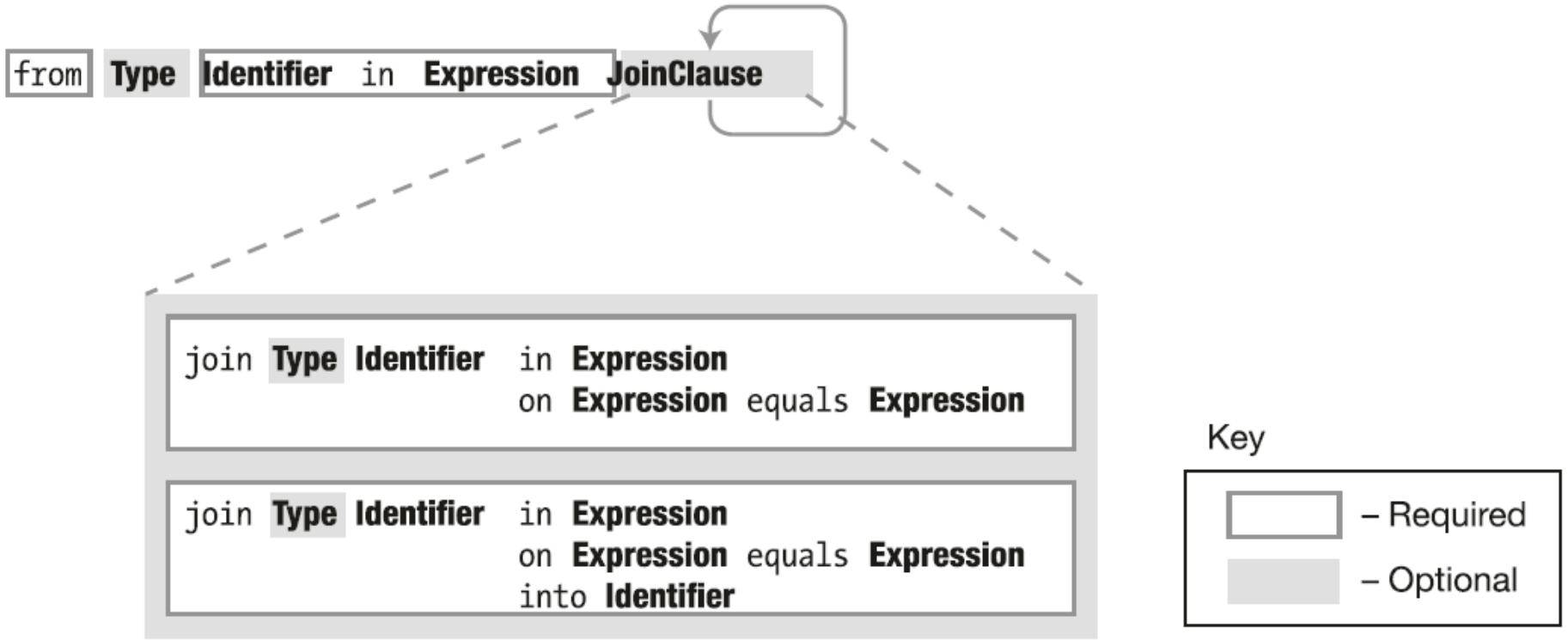
from *Type Item* in *Items*

```
int[ ] arr1 = {10, 11, 12, 13};
          переменная итерации
                          ↓
var query = from item in arr1
            where item < 13 //← использование переменной
            select item;      //← использование переменной
foreach( var item in query )
        Console.Write("{0}, ", item );
```

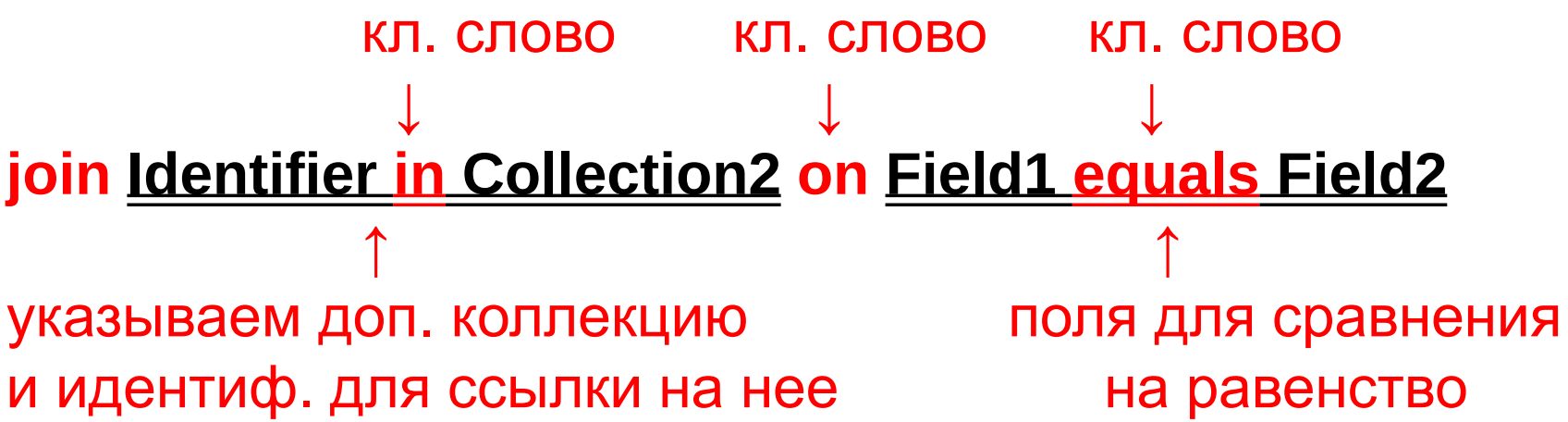
Результаты выполнения программы:

10, 11, 12,

Предложение from и соединения



Предложение join



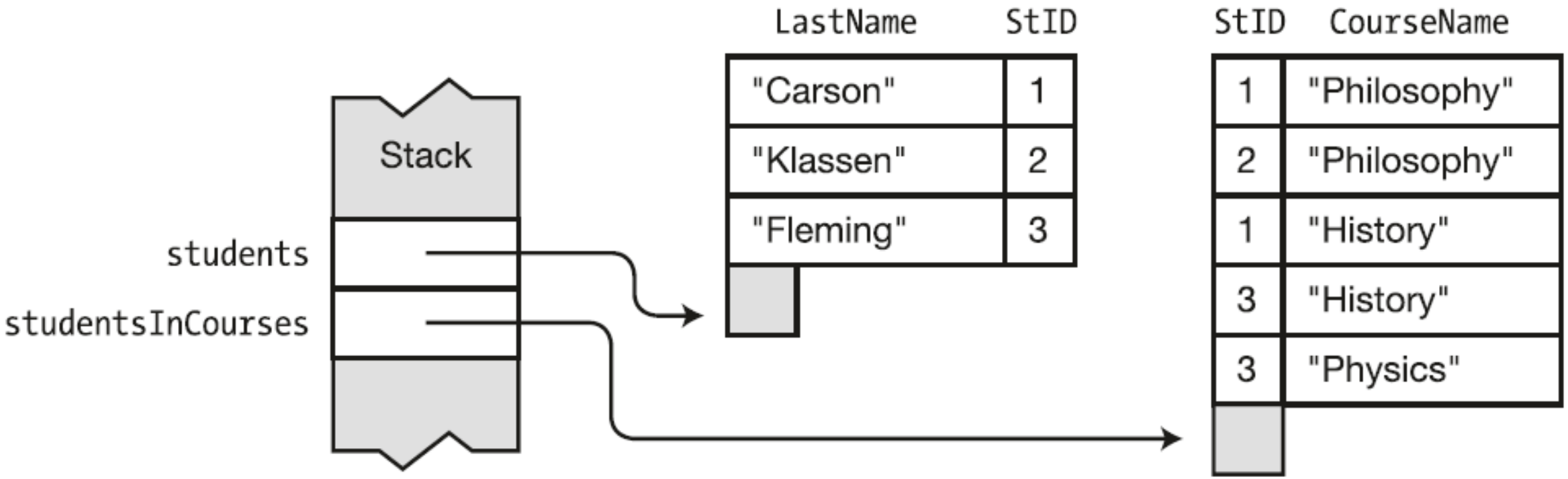

```
join Type Identifier  in Expression
                      on Expression equals Expression
```

```
join Type Identifier  in Expression
                      on Expression equals Expression
                      into Identifier
```

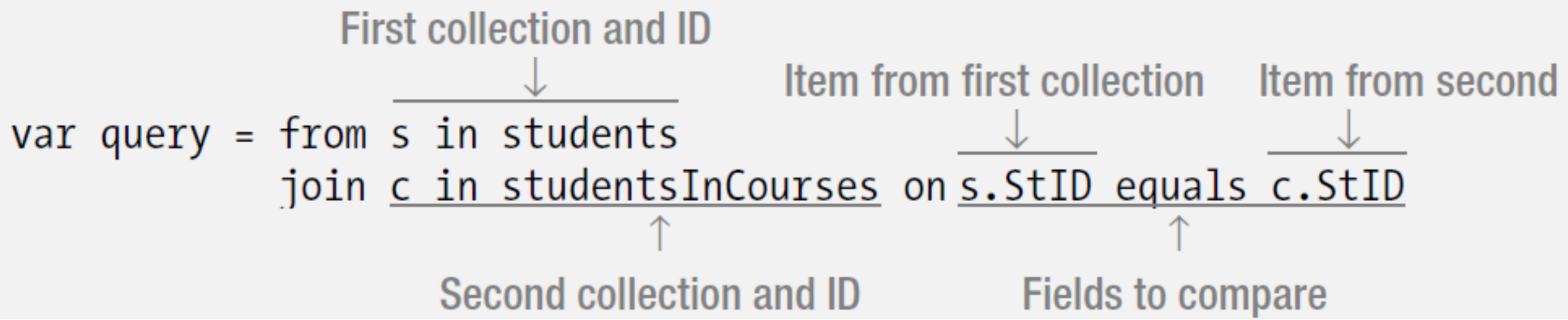
Пример классов и коллекций

```
public class Student { // Студент
    public int StID;
    public string LastName;
}
public class CourseStudent { // Дисциплина
    public string CourseName;
    public int StID;
}

static Student[] students = new Student[] {
    new Student { StID = 1, LastName = "Carson" },
    new Student { StID = 2, LastName = "Klassen" },
    new Student { StID = 3, LastName = "Fleming" },
};
static CourseStudent[] studentsInCourses = new CourseStudent[] {
    new CourseStudent { CourseName = "Art", StID = 1 }, // Carson
    new CourseStudent { CourseName = "Art", StID = 2 }, // Klassen
    new CourseStudent { CourseName = "History", StID = 1 }, // Carson
    new CourseStudent { CourseName = "History", StID = 3 }, // Fleming
    new CourseStudent { CourseName = "Physics", StID = 3 }, // Fleming
};
```

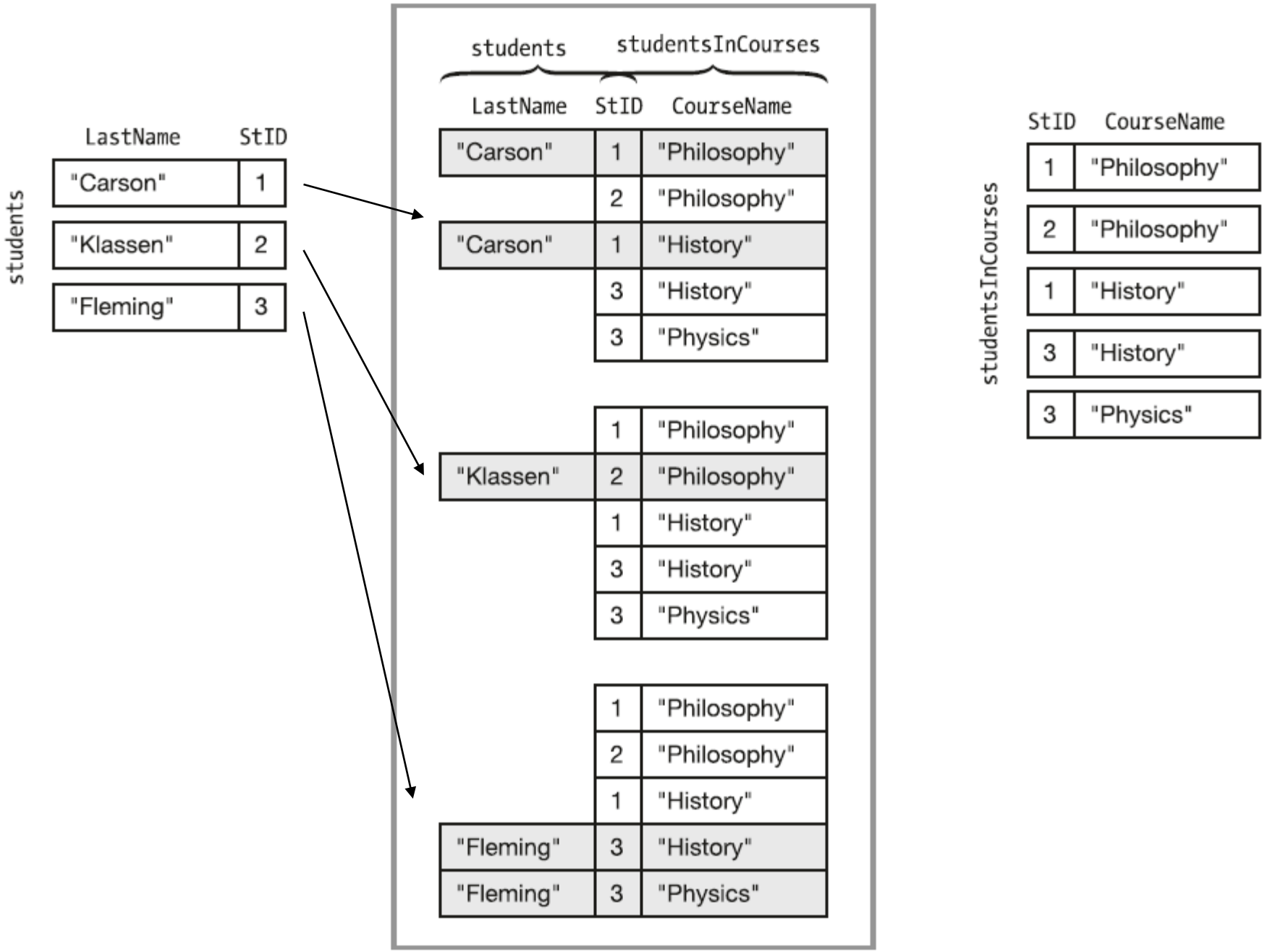
Студенты и дисциплины



Запрос с использованием join



Соединение массивов



Применение соединения

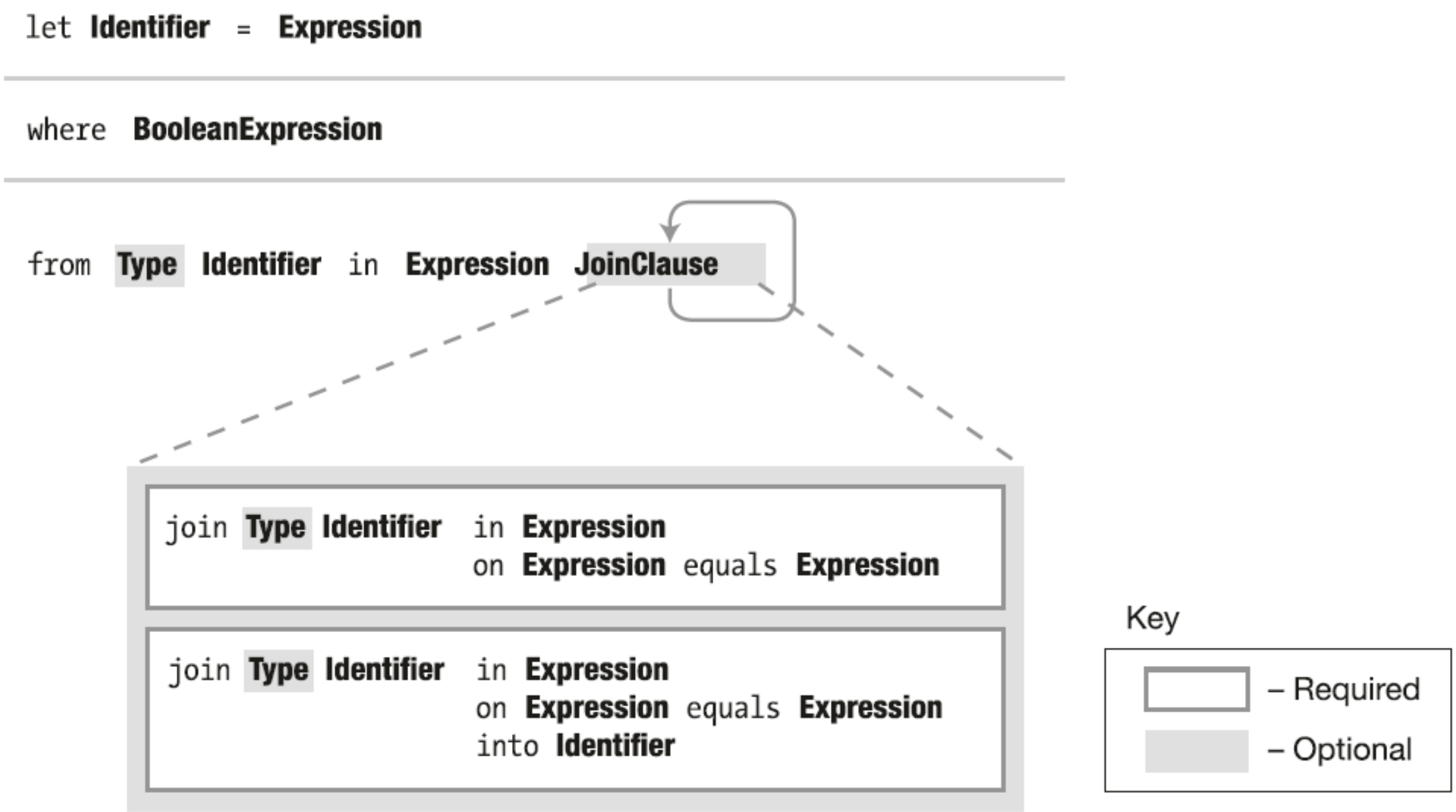
```
static void Main( )  {
   …
   // запрос: найти фамилии студентов, сдающих историю
   var query = from s in students
               join c in studentsInCourses on s.StID equals c.StID
               where c.CourseName == "History"
               select s.LastName;
   // вывести имена студентов, сдающих историю
   foreach (var q in query)
            Console.WriteLine("Student taking History: {0}", q);
}
```

Результат работы программы:

Student taking History: Carson

Student taking History: Fleming

Первая секция запроса



Предложение from и декартово произведение

```
static void Main()   {
   var groupA = new[ ] { 3, 4, 5, 6 };
   var groupB = new[ ] { 6, 7, 8, 9 };
   var someInts = from a in groupA          // первый from
      from b in groupB                      // второй from
      where a > 4 && b >= 8                 // условие-фильтр
      select new {a, b, sum = a + b}; // объект анонимного типа
   foreach (var a in someInts)
            Console.WriteLine(a);
}
```

Результат выполнения :

```
{ a = 5, b = 8, sum = 13 }
{ a = 5, b = 9, sum = 14 }
{ a = 6, b = 8, sum = 14 }
{ a = 6, b = 9, sum = 15 }
```

Предложение let (создание переменной в LINQ)

```
public static void Main()
{
    var groupA = new[] { 3, 4, 5, 6 };
    var groupB = new[] { 6, 7, 8, 9 };
    var someInts = from a in groupA
                   from b in groupB
                   let sum = a + b     // создаем переменную
                   where sum == 12
                   select new { a, b, sum };
    foreach (var a in someInts)
        Console.WriteLine(a);
}
```

Результат выполнения:

{ a = 3, b = 9, sum = 12 }
{ a = 4, b = 8, sum = 12 }
{ a = 5, b = 7, sum = 12 }
{ a = 6, b = 6, sum = 12 }

Предложение where

```
static void Main ()
{
    var groupA = new[] { 3, 4, 5, 6 };
    var groupB = new[] { 6, 7, 8, 9 };

    var someInts = from int a in groupA
                   from int b in groupB
                   let sum = a + b
                   where sum >= 11     // Условие 1
                   where a == 4        // Условие 2
                   select new { a, b, sum };

    foreach (var a in someInts)
        Console.WriteLine(a);
}
```

Результат выполнения:
{ a = 4, b = 7, sum = 11 }
{ a = 4, b = 8, sum = 12 }
{ a = 4, b = 9, sum = 13 }

Основы XML

<Имя_элемента Атрибуты> **содержание** </Имя_элемента>

<EmployeeName>Sally Jones**</EmployeeName>**

↑ ↑ ↑

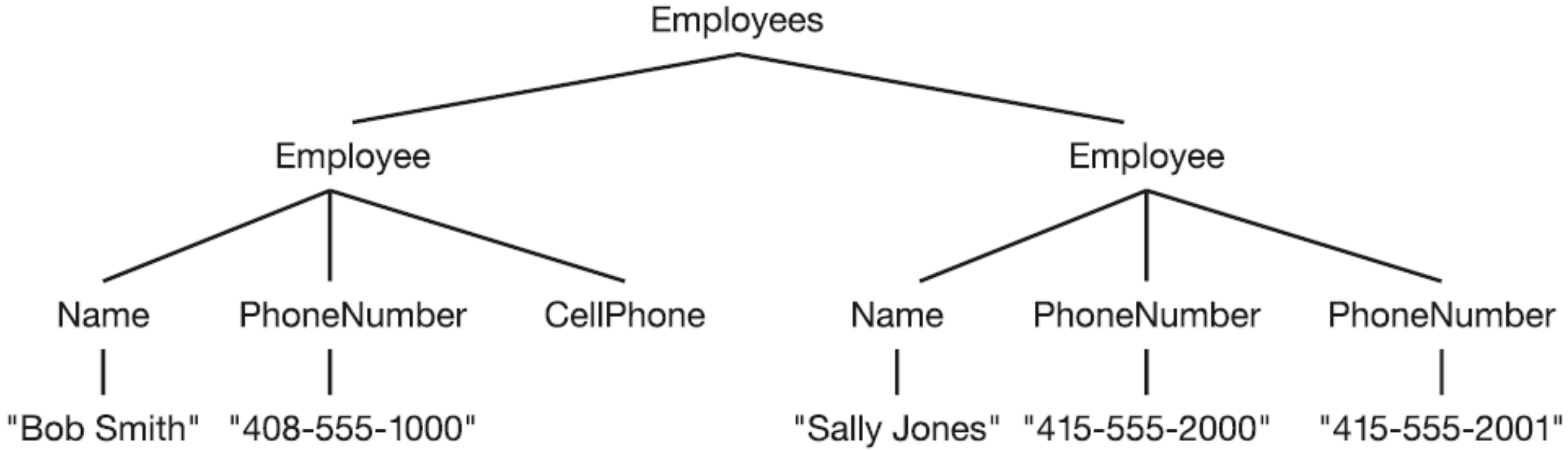
открыв. тег **содержимое** **закрывающий тег**

<PhoneNumber /> ← **узел без содержимого**

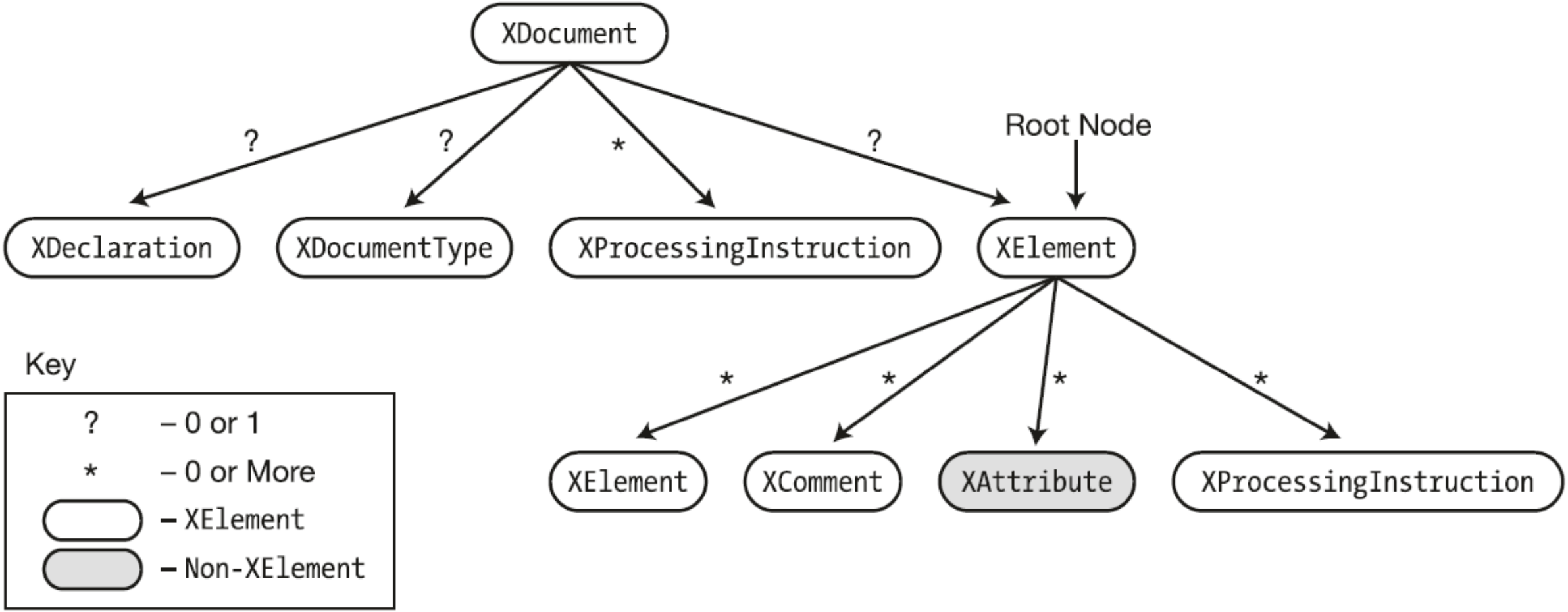
XML-документ (пример)

```
<Employees>
        <Employee>
                <Name>Bob Smith</Name>
                <PhoneNumber>408-555-1000</PhoneNumber>
                <CellPhone />
        </Employee>
        <Employee>
                <Name>Sally Jones</Name>
                <PhoneNumber>415-555-2000</PhoneNumber>
                <PhoneNumber>415-555-2001</PhoneNumber>
        </Employee>
</Employees>
```

XML-дерево (пример)



Классы LINQ to XML API



Пример работы с XML

```
using System;
using System.Xml.Linq; // требуемое пространство имен

class Program   {
static void Main( )   {
   XDocument employees1 =
      new XDocument(        // создаем XML-документ
         new XElement("Employees", // создание корневого узла
            new XElement("Name", "Bob Smith"), // создание узла
            new XElement("Name", "Sally Jones") // создание узла
       )      );
   employees1.Save("EmployeesFile.xml");     // сохранить в файл
   XDocument employees2 = XDocument.Load("EmployeesFile.xml"); // загруз.
   Console.WriteLine(employees2);                // вывести документ
}            }
```

Результат работы:

```
<Employees>
<Name>Bob Smith</Name>
<Name>Sally Jones</Name>
</Employees>
```

Создание XML дерева

```
using System;
using System.Xml.Linq; // требуемое пространство имен
class Program  {
static void Main( ) {
XDocument employeeDoc =
            new XDocument(          // создаем XML-документ
                new XElement("Employees", // создание корневого узла
                    new XElement("Employee", // первый сотрудник
                        new XElement("Name", "Bob Smith"),
                        new XElement("PhoneNumber", "408-555-1000") ),
                    new XElement("Employee", // второй сотрудник
                        new XElement("Name", "Sally Jones"),
                        new XElement("PhoneNumber", "415-555-2000"),
                        new XElement("PhoneNumber", "415-555-2001") )
            )           );
Console.WriteLine(employeeDoc); // вывести документ
}
}
```

Результат – представление XML-дерева

```
<Employees>
        <Employee>
                <Name>Bob Smith</Name>
                <PhoneNumber>408-555-1000</PhoneNumber>
        </Employee>
        <Employee>
                <Name>Sally Jones</Name>
                <PhoneNumber>415-555-2000</PhoneNumber>
                <PhoneNumber>415-555-2001</PhoneNumber>
        </Employee>
</Employees>
```

Методы LINQ to XML

Имя метода	Класс	Тип возврата	Описание
Nodes	XDocument XElement	IEnumerable<object>	Возвращает все подузлы данного узла, безотносительно к их типам
Elements	XDocument XElement	IEnumerable<XElement>	Возвращает все элементы текущего узла или все подузлы с определенным именем
Element	XDocument XElement	XElement	Возвращает первый подузел XElement текущего узла или первый подузел с определенным именем
Descendants	XElement	IEnumerable<XElement>	Возвращает все подчиненные подузлы XElement или все подчиненные подузлы XElement с определенным именем, безотносительно к их уровню вложения ниже текущего узла
DescendantsAndSelf	XElement	IEnumerable<XElement>	Делает то же, что и Descendants, но включает и текущий узел
Ancestors	XElement	IEnumerable<XElement>	Возвращает все узлы-предки типа XElement или все все узлы-предки типа XElement , размещенные выше текущего узла, которые имеют определенное имя
AncestorsAndSelf	XElement	IEnumerable<XElement>	Делает то же, что и Ancestors , но включает и текущий узел
Parent	XElement	XElement	Возвращает родительский узел текущего узла

Примеры LINQ to XML

IEnumerable<**XComment**> comments = d.Nodes().OfType<**XComment**>();
Результат – все узлы типа XComment

Elements() – упрощение для Nodes().OfType<**XElement**>()

IEnumerable<**XElement**> empPhones = emp.Elements("PhoneNumber");
Результат – все узлы типа **XElement** с названием "PhoneNumber".

Пример для employeeDoc

Get first child XElement named "Employees"

```
XElement root = employeeDoc.Element("Employees");
IEnumerable<XElement> employees = root.Elements();
foreach (XElement emp in employees)
{
```

Get first child XElement named "Name"

```
    XElement empNameNode = emp.Element("Name");
    Console.WriteLine(empNameNode.Value);

```

Get all child elements named "PhoneNumber"

```
    IEnumerable<XElement> empPhones =
    emp.Elements("PhoneNumber");
    foreach (XElement phone in empPhones)
        Console.WriteLine(" {0}", phone.Value);
}
```

Результат:

Bob Smith
408-555-1000
Sally Jones
415-555-2000
415-555-2001

Метод Add()

```
static void Main()  {
XDocument xd = new XDocument( // создаем XML
          new XElement("root",
          new XElement("first")  )
          );
Console.WriteLine("Original tree");
Console.WriteLine(xd); Console.WriteLine(); // вывод

XElement rt = xd.Element("root"); // получаем первый узел
rt.Add( new XElement("second")); // добавляем дочерний узел
rt.Add( new XElement("third"),      // добавляем еще 3 потомка
          new XComment("Important Comment"),
          new XElement("fourth"));
Console.WriteLine("Modified tree");
Console.WriteLine(xd);                  // вывод результата
 }
```

Результаты

Original tree
<root>
<first />
</root>
Modified tree
<root>
<first />
<second />
<third />
<!--Important Comment-->
<fourth />
</root>

Методы LINQ to XML

Method Name	Call From	Description
Add	Parent	Adds new child nodes after the existing child nodes of the current node
AddFirst	Parent	Adds new child nodes before the existing child nodes of the current node
AddBeforeSelf	Node	Adds new nodes before the current node at the same level
AddAfterSelf	Node	Adds new nodes after the current node at the same level
Remove	Node	Deletes the currently selected node and its contents
RemoveNodes	Node	Deletes the currently selected XElement and its contents
SetElement	Parent	Sets the contents of a node
ReplaceContent	Node	Replaces the contents of a node

Работа с атрибутами XML (1)

```
XDocument xd = new XDocument( // создание дерева XML
                        имя     значение
new XElement("root",    ↓         ↓
        new XAttribute("color", "red"),  // конструктор атрибута
        new XAttribute("size", "large"), // конструктор атрибута
        new XElement("first"),        // конструктор узла
        new XElement("second")   // конструктор узла
)
);
Console.WriteLine(xd);
```

Результаты:

```
<root color="red" size="large">
<first />
<second />
</root>
```

Работа с атрибутами XML (2)

```
static void Main( )  {
XDocument xd = new XDocument( // создание дерева XML
        new XElement("root",
        new XAttribute("color", "red"),
        new XAttribute("size", "large"),
        new XElement("first")
        )
        );
Console.WriteLine(xd); Console.WriteLine();     // вывод дерева XML
XElement rt = xd.Element("root");               // получаем узел
XAttribute color = rt.Attribute("color");       // получаем атрибут
XAttribute size = rt.Attribute("size");         // получаем атрибут
Console.WriteLine("color is {0}", color.Value); // вывод значения атрибута
Console.WriteLine("size is {0}", size.Value);   // вывод значения атрибута
}
```

Работа с атрибутами XML. Удаление

```
static void Main( ) {
   XDocument xd = new XDocument(
          new XElement("root",
             new XAttribute("color", "red"),
             new XAttribute("size", "large"),
             new XElement("first")
             )
          );
   XElement rt = xd.Element("root");          // получаем узел
   rt.Attribute("color").Remove();            // удаляем атрибут color
   rt.SetAttributeValue("size", null);        // удаляем атрибут size
   Console.WriteLine(xd);
}
```

Метод SetAttributeValue()

```
static void Main( ) {
XDocument xd = new XDocument(
new XElement("root",
        new XAttribute("color", "red"),
        new XAttribute("size", "large"),
        new XElement("first")));
XElement rt = xd.Element("root");          // получаем узел
rt.SetAttributeValue("size", "medium");  // изменяем значение атрибута
rt.SetAttributeValue("width", "narrow");  // добавляем атрибут
Console.WriteLine(xd); Console.WriteLine();
}
```

Результат:

```
<root color="red" size="medium" width="narrow">
<first />
</root>
```

Другие типы узлов

XComment:

```
new XComment("This is a comment")
```

XDeclaration:

```
new XDeclaration("1.0", "utf-8", "yes")
```

XProcessingInstruction:

```
new XProcessingInstruction( "xml-stylesheet",
        @"href=""stories""", type=""text/css""")
```

XDeclaration, XComment, XProcessingInstruction

```
static void Main( )  {
XDocument xd = new XDocument(
        new XDeclaration("1.0", "utf-8", "yes"),
        new XComment("This is a comment"),
        new XProcessingInstruction("xml-stylesheet",
                @"href=""stories.css"" type=""text/css"""),
        new XElement("root",
        new XElement("first"),
        new XElement("second")
        )  );
}
```

Результаты построения XML

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!--This is a comment-->
<?xml-stylesheet href="stories.css" type="text/css"?>
<root>
<first />
<second />
</root>
```

Построение XML дерева

```
static void Main()    {
        XDocument xd = new XDocument(
        new XElement("MyElements",
            new XElement("first",
                new XAttribute("color", "red"),
                new XAttribute("size", "small")),
            new XElement("second",
                new XAttribute("color", "red"),
                new XAttribute("size", "medium")),
            new XElement("third",
                new XAttribute("color", "blue"),
                new XAttribute("size", "large"))));
        Console.WriteLine(xd); // Display XML tree
        xd.Save("SimpleSample.xml"); // Save XML tree
    }
```

Результаты построения XML

Результаты на консоли:

```
<MyElements>
  <first color="red" size="small" />
  <second color="red" size="medium" />
  <third color="blue" size="large" />
</MyElements>
```

Результаты в файле:

```
<?xml version="1.0" encoding="utf-8"?>
<MyElements>
  <first color="red" size="small" />
  <second color="red" size="medium" />
  <third color="blue" size="large" />
</MyElements>
```

Использование LINQ-запросов с LINQ to XML

```
static void Main( ) {
XDocument xd = XDocument.Load("SimpleSample.xml"); // загрузка
XElement rt = xd.Element("MyElements"); // получаем корень

var xyz = from e in rt.Elements()            // выбираем узлы,
    where e.Name.ToString().Length == 5   // имена которых из 5 символов
    select e;

foreach (XElement x in xyz)                          // отображаем выбранные
    Console.WriteLine(x.Name.ToString());          // элементы
Console.WriteLine();

foreach (XElement x in xyz)
    Console.WriteLine("Name: {0}, color: {1}, size: {2}",
            x.Name,
            x.Attribute("color").Value,
            x.Attribute("size") .Value);
}
```

Результаты

Файл SimpleSample.xml расположен в папке с *.exe

Получены следующие результаты:
first
third

Name: first, color: red, size: small
Name: third, color: blue, size: large

Использование LINQ-запросов с LINQ to XML

```
static void Main( )  {
XDocument xd = XDocument.Load("SimpleSample.xml"); XElement rt =
xd.Element("MyElements");
var xyz = from e in rt.Elements()
select new { e.Name, color = e.Attribute("color") };
foreach (var x in xyz)
    Console.WriteLine(x); // форматирование по умолчанию
Console.WriteLine();

foreach (var x in xyz)
Console.WriteLine("{0,-6}, color: {1, -7}", x.Name,                x.color.Value);
}
```

Результаты

Файл SimpleSample.xml расположен в папке с *.exe

Получены следующие результаты:

{ Name = first, color = color="red" }
{ Name = second, color = color="red" }
{ Name = third, color = color="blue" }

first , color: red
second, color: red
third , color: blue