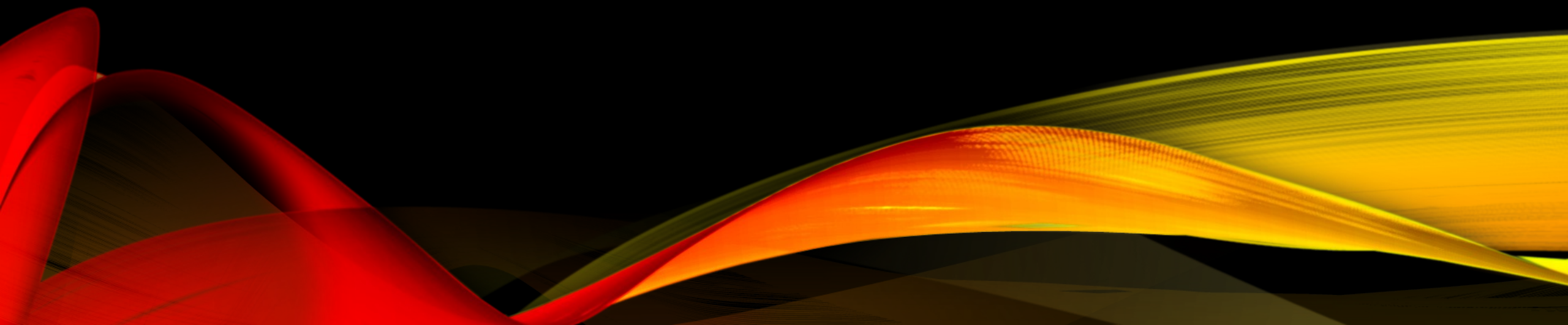
ЛЕКЦИЯ 15-16

- Модуль 3
- 28.02.2024
- LINQ

ЦЕЛИ ЛЕКЦИИ

- Изучить анонимный тип в программах на C#
- Познакомиться с технологиями LINQ
- Разобраться с вариантами синтаксиса LINQ-запросов

АНОНИМНЫЕ ТИПЫ



АНОНИМНЫЕ ТИПЫ

Анонимный тип – создаваемый компилятором ссылочный тип (прямой наследник object), инкапсулирующий свойства только для чтения

- Инкапсулирует свойства только для чтения в один объект без необходимости предварительного создания типа; другие члены класса недопустимы
  - Методов и событий в анонимном типе быть не может!
- Имя типа известно на уровне компилятора и не доступно на уровне исходного кода
- Тип каждого свойства (только для чтения) выводится компилятором

АНОНИМНЫЕ ТИПЫ

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 class Student // Студент.
{
    public int StID { get; set; }
    public string LastName { get; set; }
}
```

```
public class CourseStudent // Дисциплина.
{
    public string CourseName { get; set; }
    public int StID { get; set; }
}
```

```
Student st = new Student { StID = 1, LastName="Ivanov" };
CourseStudent course = new CourseStudent { CourseName = "Programming", StID = 1 };

// Обязательно используем анонимый тип для идентификатора allInfo.
var allInfo = new { ID = st.StID, LastName = st.LastName, Course = course.CourseName };
Console.WriteLine(allInfo);
```

Создание анонимного типа с использованием **new** и инициализатора

ОБ ЭКВИВАЛЕНТНОСТИ ОБЪЯВЛЕНИЙ

```
var allInfo = new { st.StID, st.LastName, course.CourseName };
```

равносильно

```
var allInfo = new { ID = st.StID, LastName = st.LastName, Course = course.CourseName };
```

Какими будут идентификаторы полей при таких вариантах создания и инициализации объекта анонимного типа?

ВЫВОД ОБЪЕКТА АНОНИМНОГО ТИПА

```
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 }
```

LINQ

- **L**anguage **IN**tegrated **Q**uery
- LINQ – методология, упрощающая и унифицирующая реализацию доступа к данным разных видов
- Технологии LINQ – это языковые конструкции, обеспечивающие согласованное функционирование запросов для объектов, реляционных БД и XML

Кроме того, LINQ-запрос может написать **к любой коллекции**, поддерживающей интерфейс IEnumerable / IEnumerable<T>

РЕАЛИЗАЦИИ LINQ ДЛЯ РАЗНЫХ ИСТОЧНИКОВ ДАННЫХ

- LINQ to Objects (https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/linq/linq-to-objects)
- LINQ to ADO.NET (https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/linq/linq-to-adonet-portal-page)
  - LINQ to Entities (https://learn.microsoft.com/ru-ru/dotnet/framework/data/adonet/ef/language-reference/linq-to-entities)
  - LINQ to SQL (https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/sql/linq/)
  - LINQ to DataSet (https://learn.microsoft.com/ru-ru/dotnet/framework/data/adonet/queries-in-linq-to-dataset)
- LINQ to XML (https://learn.microsoft.com/ru-ru/dotnet/standard/linq/linq-xml-overview)
- LINQ to JSON (JSON.NET, https://www.newtonsoft.com/json/help/html/LINQtoJSON.htm)

ПРИМЕР ЗАПРОСА LINQ

```
int[] numbers = { 2, 12, 5, 15 };
// Определение запроса.
IEnumerable<int> lowNums =
                    from n in numbers
                    where n < 10
                    select n;
// Выполнение запроса. Вариант 1.
foreach (var x in lowNums) Console.Write($"{x} ");
   Console.WriteLine();
// Выполнение запроса. Вариант 2.
lowNums.ToList().ForEach(ln => Console.Write($"{ln} "));
```

Вывод. Вариант 1

```
2 5
```

Вывод. Вариант 2

```
2 5
```

КОНТЕКСТНО-ЗАВИСИМЫЕ СЛУЖЕБНЫЕ СЛОВА ЯЗЫКА LINQ

from
where
select
group
into
orderby
join
let

Контекстно-зависимые ключевые слова
in
on
equals
by
ascending
descending

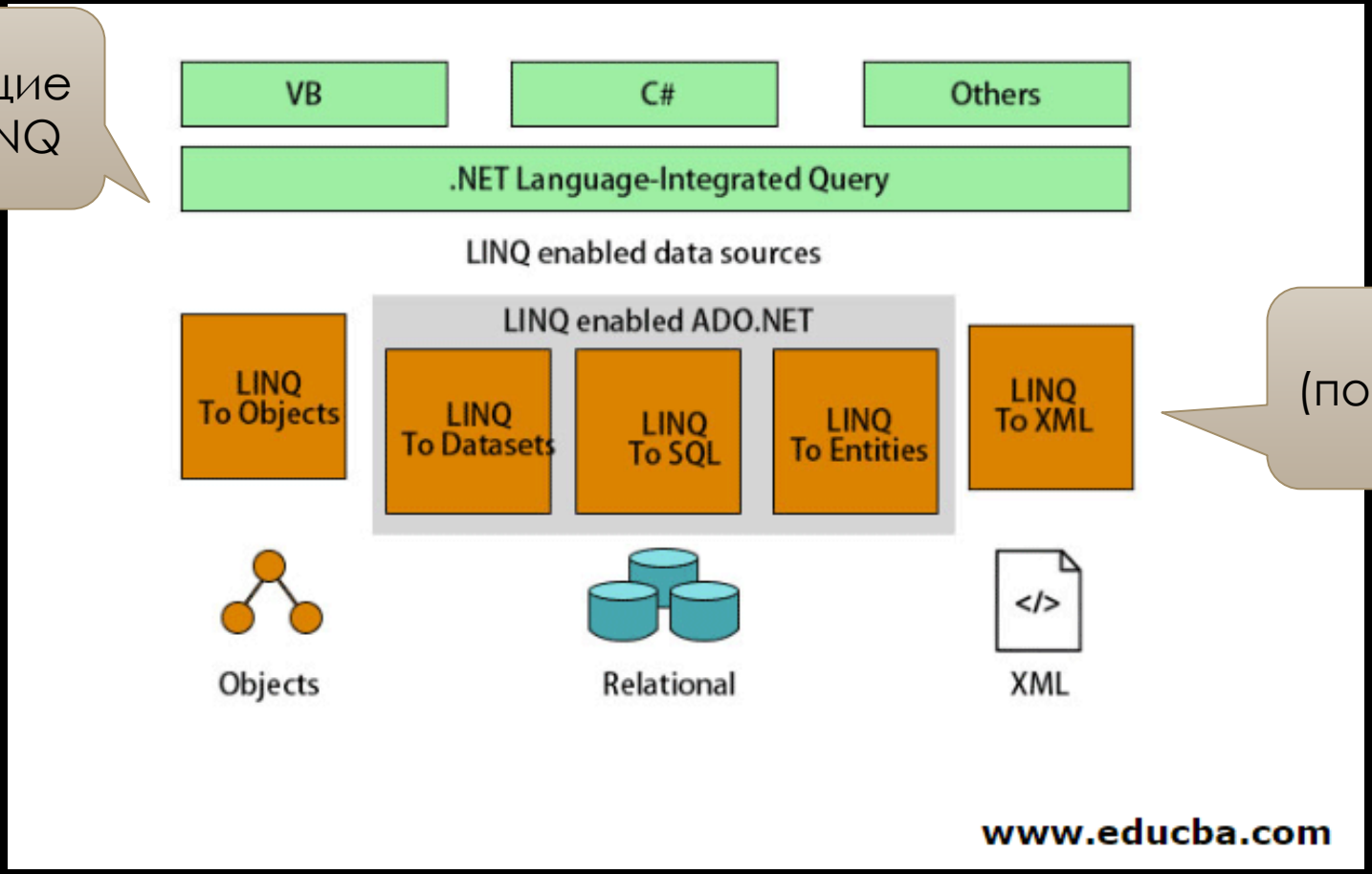
JOIN

GROUP

ORDERBY

ПРОВАЙДЕРЫ LINQ

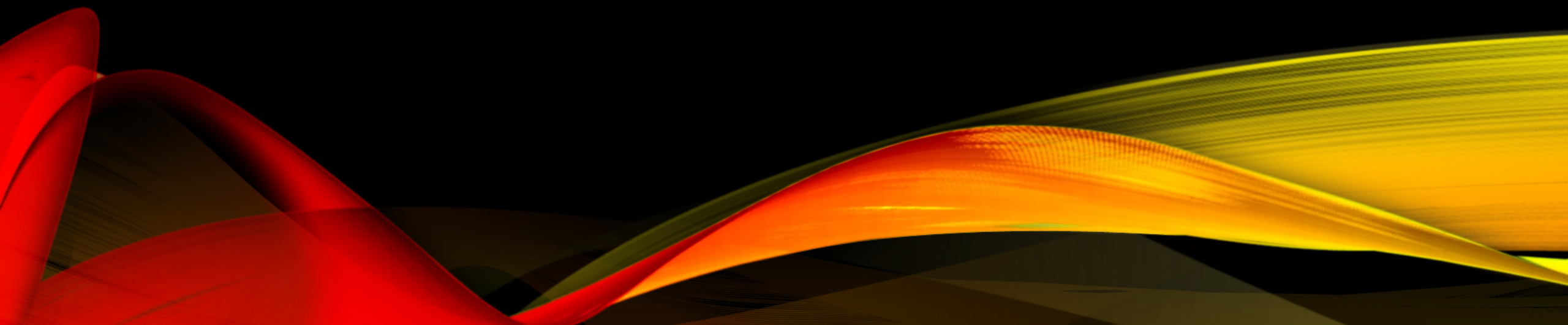
Языки, допускающие использование LINQ



Провайдеры (поставщики) данных LINQ

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

СИНТАКСИС МЕТОДОВ ДЛЯ LINQ



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

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) теоретико-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.

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

ДЕЛЕГАТЫ В КАЧЕСТВЕ ПАРАМЕТРОВ

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

↑
обобщеный делегат

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

Результат работы программы:

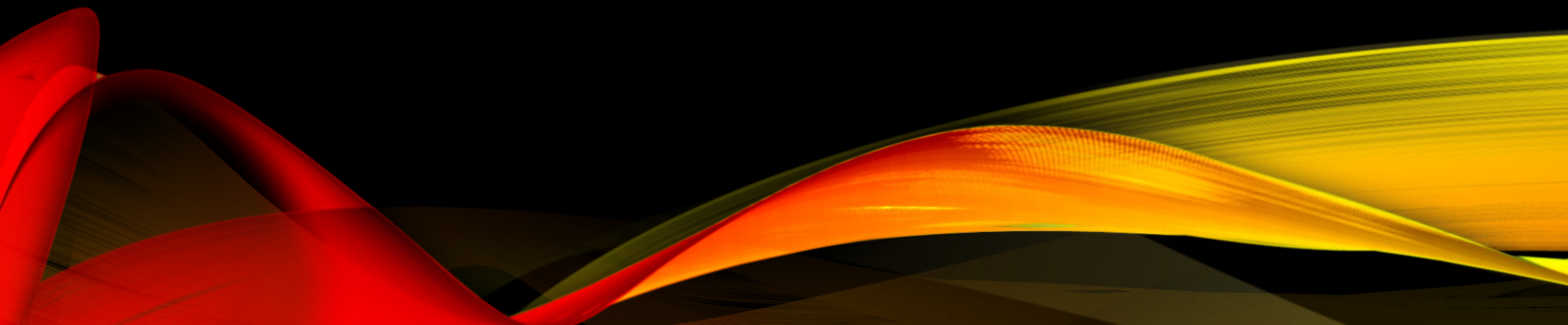
```
Count of odd numbers: 4
```

ПРИМЕР С ДЕЛЕГАТОМ

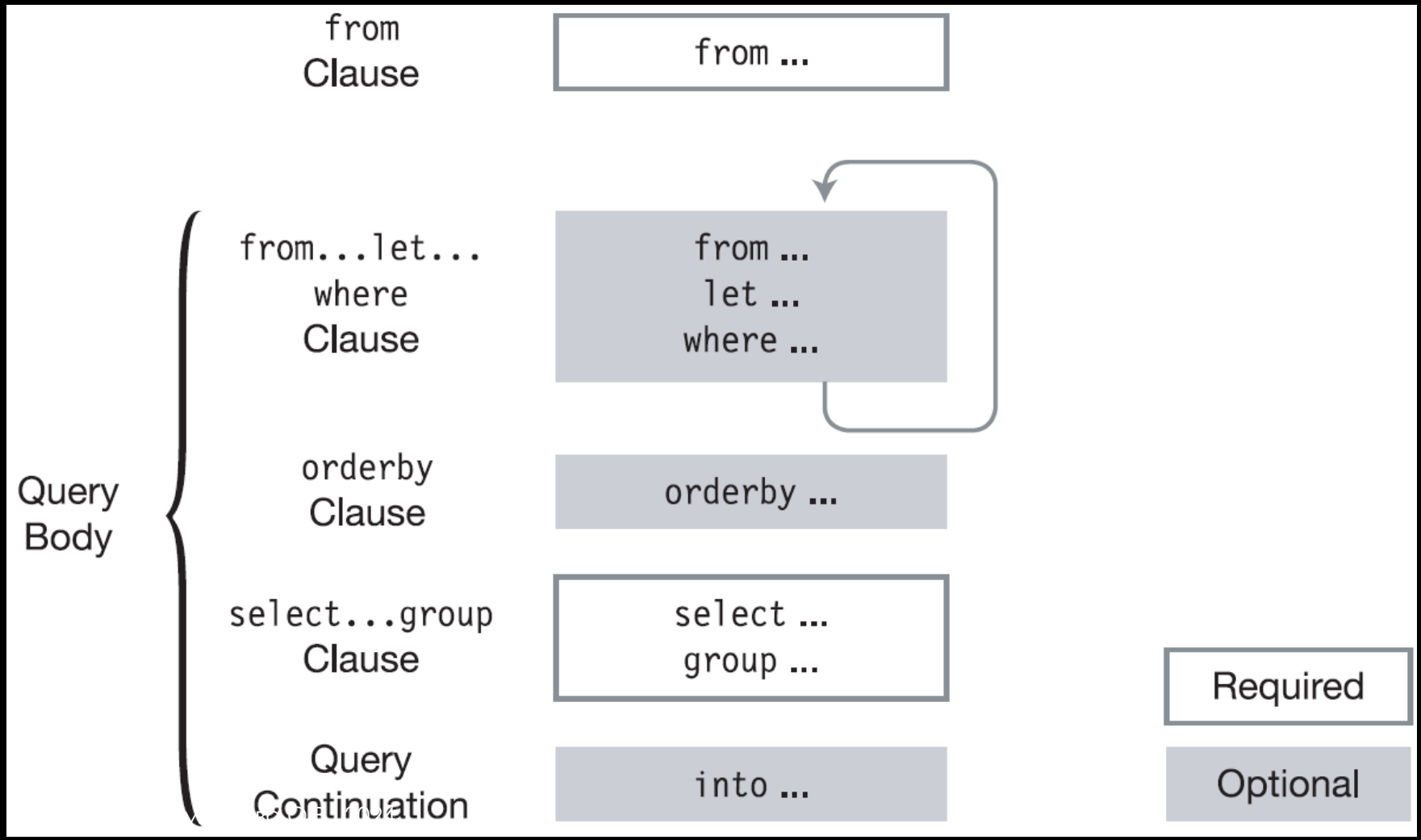
```
internal class Program
{
   static bool IsOdd(int x) // Метод для передачи через делегат.
   { return x % 2 == 1; }   // true для нечетных.

   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); // Используем делегат.
      Console.WriteLine($"Кол-во нечетных: {countOdd}") ;
   }
}
```

Результат работы программы:
4

СИНТАКСИС ЗАПРОСОВ ДЛЯ LINQ



СТРУКТУРА ВЫРАЖЕНИЙ В ЗАПРОСАХ



СИНТАКСИС ЗАПРОСОВ И МЕТОДОВ

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

// 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();
```

```
Console.WriteLine("Query::");
numsQuery.ToList().ForEach(x => Console.Write($"{x} ") ); // 2 5 17 16
Console.WriteLine();
Console.WriteLine("Method::");
numsMethod.ToList().ForEach(x => Console.Write($"{x} ")); // 2 5 17 16
Console.WriteLine();
Console.WriteLine("Combine::");
Console.WriteLine($"numsCount = {numsCount}");
```

ПЕРЕМЕННЫЕ ЗАПРОСОВ

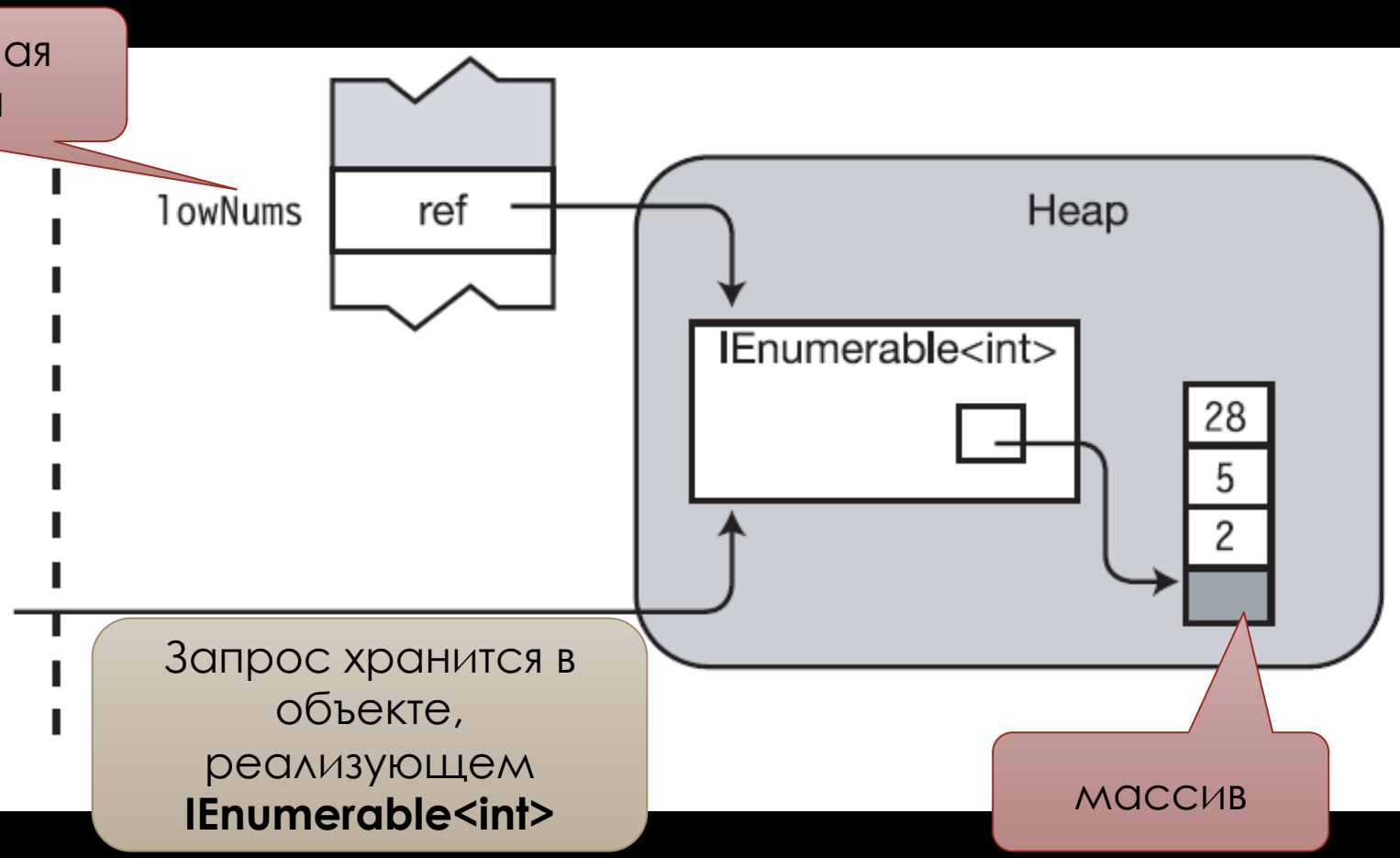
```
int[] numbers = { 2, 5, 28 };

// 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.
```

Вывод:

```
numsCount = 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


```
Point[] points = { new Point(2, 2),
        new Point (-2, 1),
        new Point(1, 1),
        new Point (0, -2),
        new Point(3, 3)
        };

var sortedPoints = from s in points where s.X>0 && s.Y>0
                   orderby s.X
                   select s;
sortedPoints.ToList().ForEach(x => Console.WriteLine($"{x}"));
```

```
public struct Point {
    public int X { get; set; }
    public int Y { get; set; }
    public Point(int x,int y) =>
          (X,Y)= (x,y);

    public override string ToString() =>
          $"{X}:{Y}";
}
```

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

```
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" }
      };
// Используем анонимный тип select new { s.LastName, s.FirstName, s.Major };

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

Вывод:

```
Jones
Smith
Fleming
```

ИСПОЛЬЗОВАНИЕ GROUP

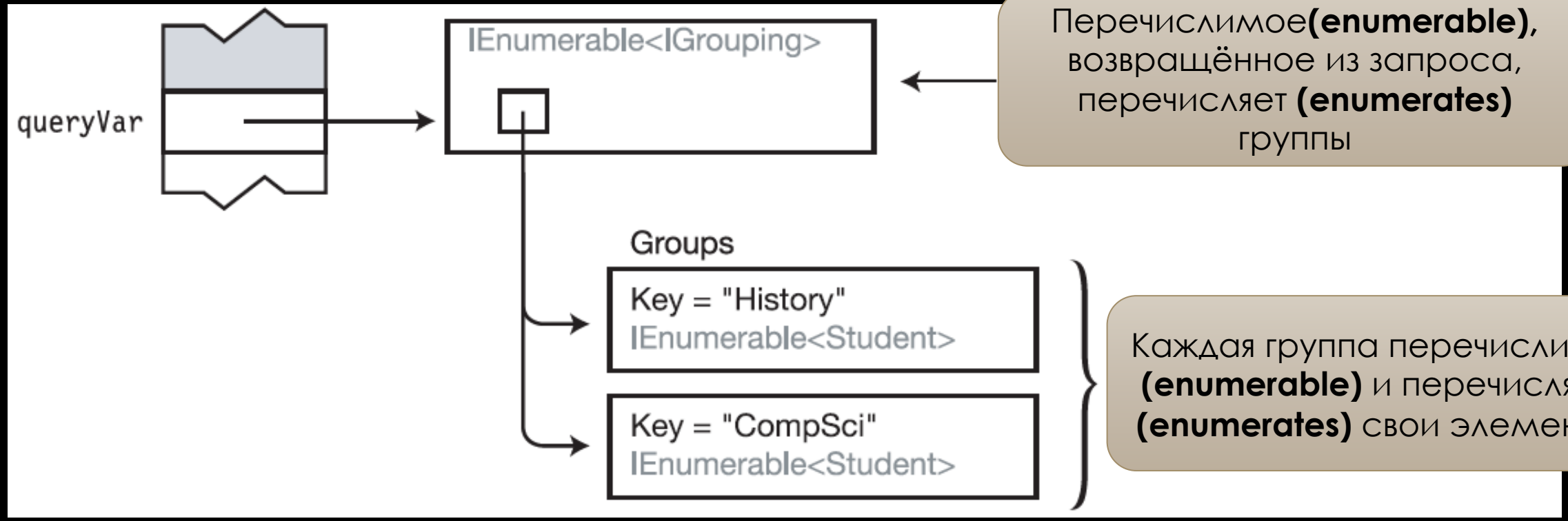
```
group student by student.Major;
     ↑           ↑
кл. слово     кл. слово
```

Вывод:

```
Ключ группы History
        Jones, Mary
        Fleming, Carol
Ключ группы CompSci
        Smith, Bob
```

```
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 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}");
}
```

СХЕМА РАБОТЫ GROUP



Перечислимое**(enumerable)**, возвращённое из запроса, перечисляет **(enumerates)** группы

Каждая группа перечислима **(enumerable)** и перечисляет **(enumerates)** свои элементы

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

ПРОДОЛЖЕНИЕ ЗАПРОСА

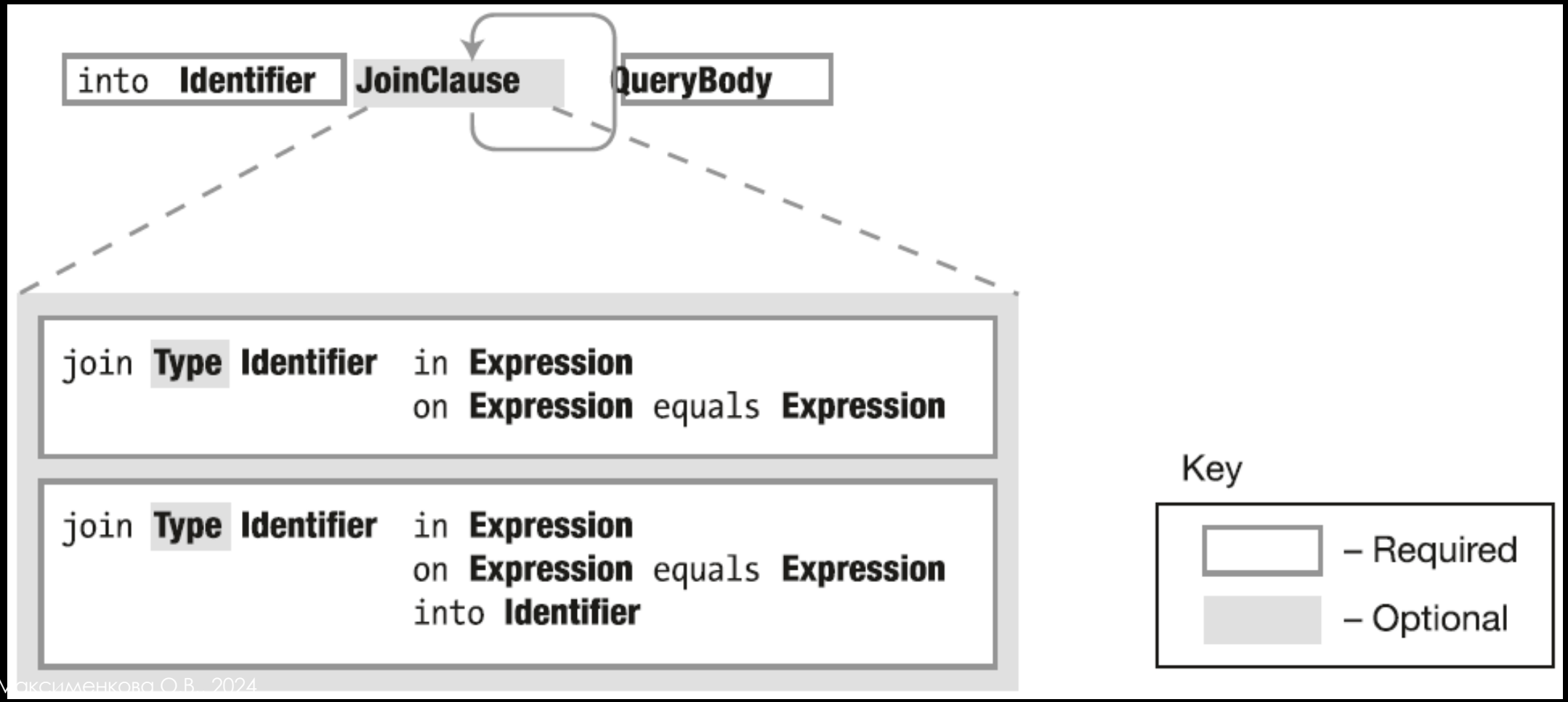
```
var groupA = new[] { 3, 4, 4, 5, 6 };
var groupB = new[] { 4, 4, 5, 6, 7 };
var someInts = from a in groupA
               join b in groupB on a equals b
               into groupAandB  // Продолжение запроса
               from c in groupAandB
               select c;

someInts.ToList().ForEach(x => Console.Write($"{x} "));
```

Создан временный идентификатор для сохранения результата join

Вывод:

4 4 4 4 5 6

ПРОДОЛЖЕНИЕ ЗАПРОСА (СХЕМА)



ПРЕДЛОЖЕНИЕ FROM

объявление переменной итерации

↓

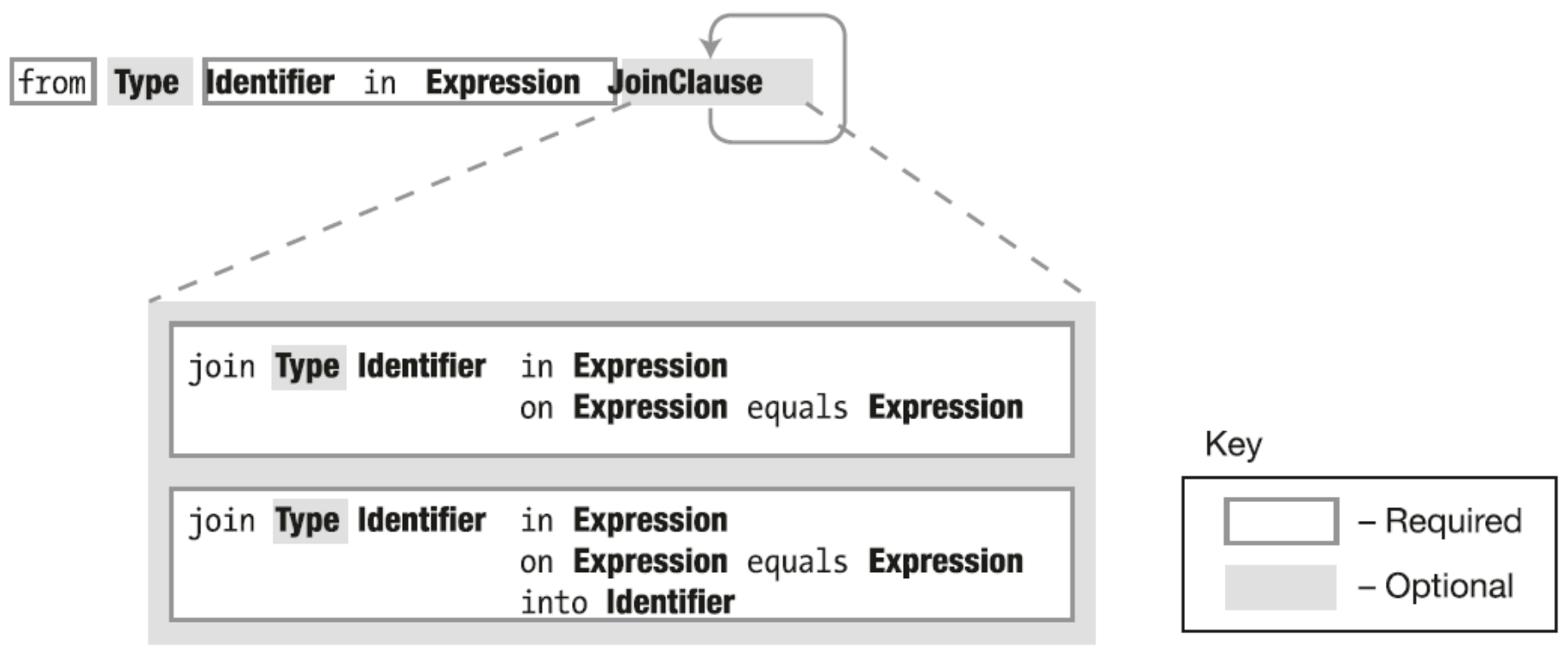
```
from Type Item in Items
```

```
int[] arr1 = { 10, 11, 12, 13 };
// item -  переменная итерации.
var query = from item in arr1
            where item < 13 //  использование переменной
            select item;    //  использование переменной
query.ToList().ForEach(x => Console.Write($"{x} "));
```

Результаты выполнения программы:

```
10 11 12
```

ПРЕДЛОЖЕНИЕ FROM И СОЕДИНЕНИЯ



ПРЕДЛОЖЕНИЕ JOIN

кл. слово кл. слово кл. слово

```
join Identifier in Collection2 on Field1 equals Field2
```

указываем доп. коллекцию и идентиф. для ссылки на нее

поля для сравнения на равенство

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

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

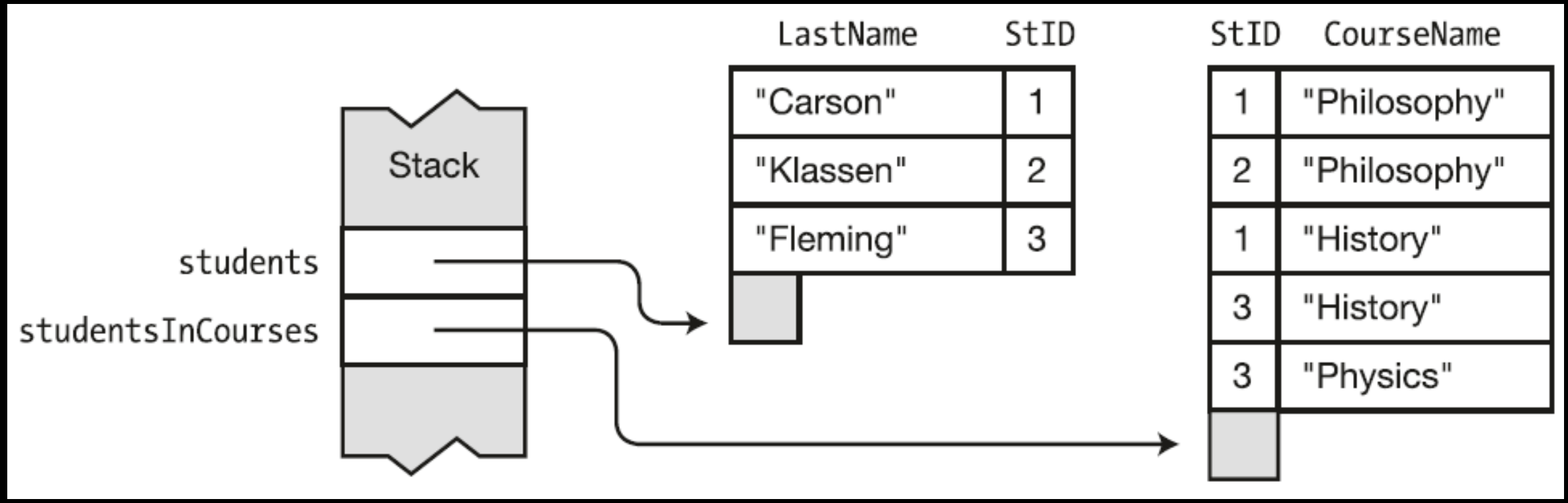
ПРИМЕР КЛАССОВ И КОЛЛЕКЦИЙ

```
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



ФРАГМЕНТ ЗАПРОСА С ИСПОЛЬЗОВАНИЕМ JOIN

```
from s in students // Первая коллекция и введённый идентификатор s
join c in studentsInCourses // Вторая коллекция и введённый идентификатор c
on s.StID equals c.StID // Сравнение элемента данных первой коллекции с элементом данных второй коллекции
```

СОЕДИНЕНИЕ МАССИВОВ

students

LastName	StID
"Carson"	1
"Klassen"	2
"Fleming"	3

studentsInCourses

StID	CourseName
1	"Philosophy"
2	"Philosophy"
1	"History"
3	"History"
3	"Physics"

students × studentsInCourses

LastName	StID	CourseName
"Carson"	1	"Philosophy"
	2	"Philosophy"
"Carson"	1	"History"
	3	"History"
	3	"Physics"
	1	"Philosophy"
"Klassen"	2	"Philosophy"
	1	"History"
	3	"History"
	3	"Physics"
	1	"Philosophy"
	2	"Philosophy"
	1	"History"
"Fleming"	3	"History"
"Fleming"	3	"Physics"

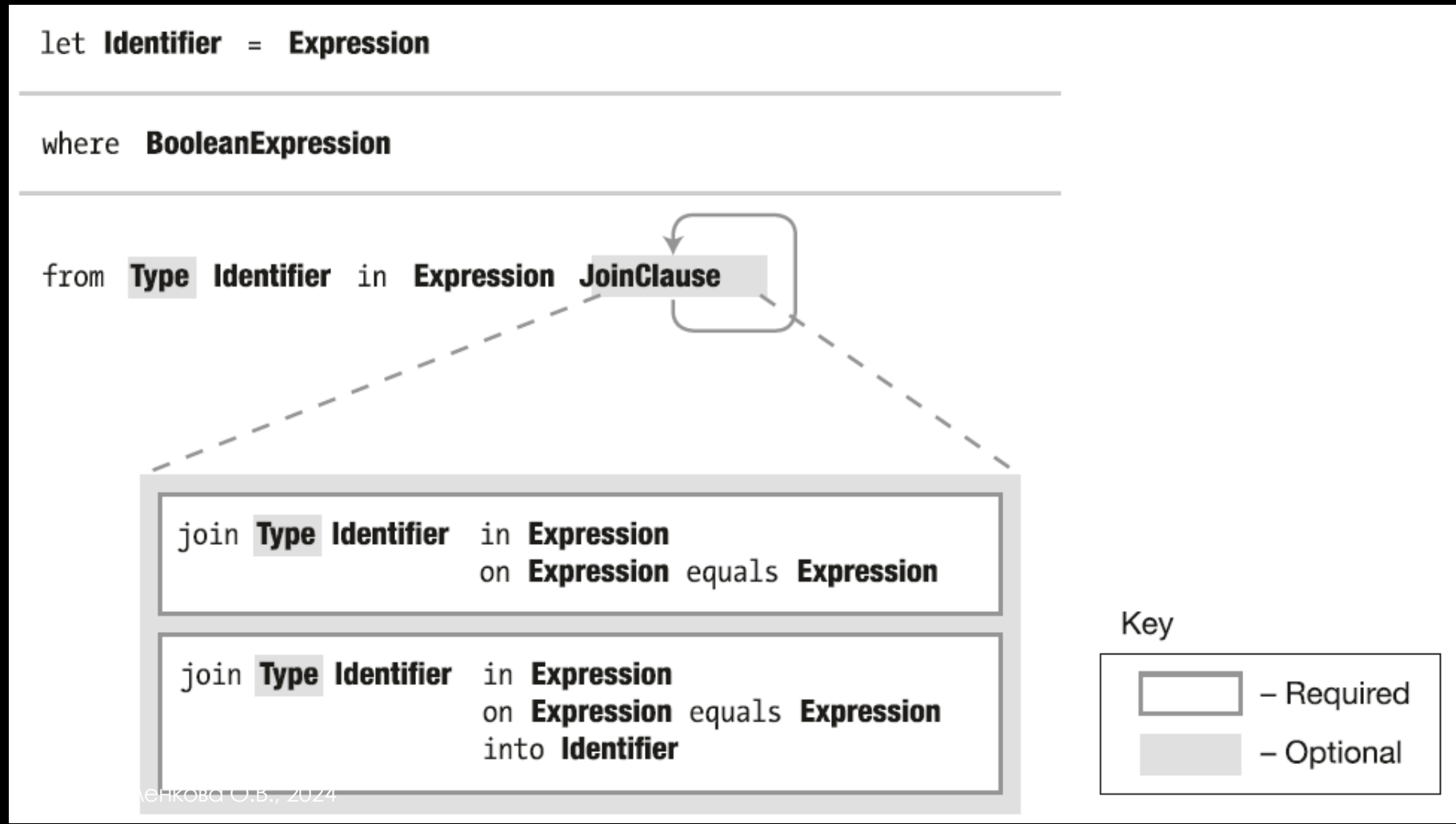
ПРИМЕНЕНИЕ СОЕДИНЕНИЯ

```
var query = from s in students
            join c in studentsInCourses
            on s.StID equals c.StID
            where c.CourseName == "History"
            select s.LastName;

query.ToList().ForEach(s => Console.WriteLine($"Studetnt taking History {s}"));
```

Результат работы программы:

```
Student taking History: Carson
Student taking History: Fleming
```

ПЕРВАЯ СЕКЦИЯ ЗАПРОСА



ПРЕДЛОЖЕНИЕ FROM И ДЕКАРТОВО ПРОИЗВЕДЕНИЕ

```
var groupA = new[] { 3, 4, 5, 6 };
var groupB = new[] { 6, 7, 8, 9 };
var someInts = from a in groupA          // Первая коллекция.
               from b in groupB          // Вторая коллекция.
               where a > 4 && b >= 8     // Условие-фильтр.
               select new { a, b, sum = a + b }; // Объект анонимного типа

someInts.ToList().ForEach(s => Console.WriteLine($"{s} "));
```

```
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)

```
var groupA = new[] { 3, 4, 5, 6 };
var groupB = new[] { 6, 7, 8, 9 };
var someInts = from a in groupA          // Первая коллекция.
        from b in groupB                 // Вторая коллекция.
        let sum = a + b                  // Создана перменная для суммы.
        where sum == 12                  // Условие-фильтр.
        select new { a, b, sum = a + b }; // Объект анонимного типа

    someInts.ToList().ForEach(s => Console.WriteLine($"{s} "));
```

```
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

```
var groupA = new[] { 3, 4, 5, 6 };
var groupB = new[] { 6, 7, 8, 9 };
var someInts = from a in groupA          // Первая коллекция.
        from b in groupB                 // Вторая коллекция.
        let sum = a + b                  // Создана переменная для суммы.
        where sum >= 12                  // Условие первое.
        where a == 4                     // Условие второе.
        select new { a, b, sum = a + b }; // Объект анонимного типа

someInts.ToList().ForEach(s => Console.WriteLine($"{s} "));
```

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

СМЕШЕНИЕ СИНТАКСИСА ЗАПРОСОВ И ВЫЗОВОВ МЕТОДОВ

```
var numbers = new int[] { 2, 6, 4, 8, 10 };

int less7Numb = (from n in numbers
                 where n < 7
                 select n).Count();

Console.WriteLine($"Count: {less7Numb}");
```

```
Результат работы программы:
Count: 3
```

ССЫЛКИ

- Ключевые слова запроса (Справочник по C#) (https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/query-keywords)
- Анонимные типы (https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/types/anonymous-types)
- Пошаговое руководство. Написание запросов на C# (https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/linq/walkthrough-writing-queries-linq)