

В.В. Подбельский

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

04 Часть 1

Процедурное Программирование на C#

Процедурное Программирование на C#

```
// using-директивы
using System;

class Program // Класс приложения
{
    static void Main() // Метод Main - точка входа
    {
        // Код решения задачи
    } // Main()
} // class Program
```

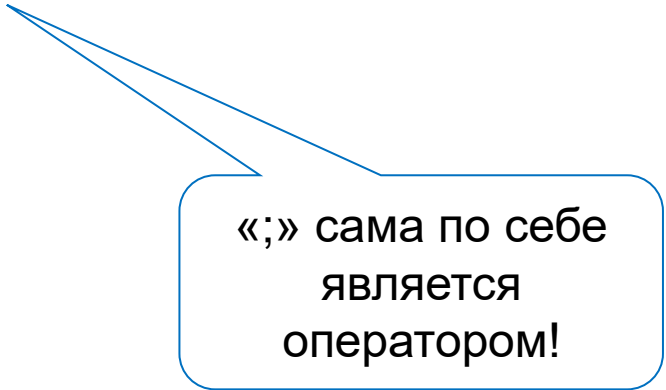
Вспомним, как выглядела простейшая программа на C#:

- Один файл;
- Один класс, объявлен явно или сгенерирован компилятором (C# 9.0).

Простые Операторы (statements)

Примеры операторов:

```
int group = 5;  
System.Console.WriteLine("Очередной вывод на экран");  
while (group++ < 2021) ;  
group++;
```



«;» сама по себе
является
оператором!

Оператор – инструкция исходного кода программы, описывающая тип или вызывающая некоторое действие программы. Оператор завершается точкой с запятой.

Блок и Составной Оператор

Блок – последовательность из нуля или более операторов, ограниченная фигурными скобками.

```
{                                // Это блок:
    int group = 5;
    System.Console.WriteLine($"Номер группы: {group}");
}
int x = 4, z = 3, d;
{                                // Это составной оператор:
    d = x;
    x = z;
    z = d;
}
```

В языках, предшествовавших С#, кроме блока определялся составной оператор – заключенная в фигурные скобки последовательность операторов, не содержащая объявлений переменных. В языке С# не стали различать эти два понятия.

Повторение: Комментарии в Коде

- Однострочные:

// Это однострочный комментарий

- С двумя ограничителями:

```
int b, /* начало */ e = 1; /* конец */
```

- Документирующие:

```
/// <summary>
```

```
/// Этот текст будет содержимым
```

```
/// элемента XML-документа
```

```
/// </summary>
```

Повторение: Простейшее Объявление Класса

```
class MyClass    // Идентификатор (имя) класса.  
{               // Начало тела класса.  
  
    // Объявления членов класса...  
  
}               // Конец тела класса.
```

Обратите внимание: В отличие от C++, в C# не требуется символ «;» после тела класса в обязательном порядке.

Объявление Класса (Declaration)

атрибуты_{опционально}

модификаторы класса_{опционально}

partial_{опционально}

ключевое слово *class*

идентификатор (имя класса)

список типизирующих параметров (в треугольных скобках)_{опционально}

родительский класс_{опционально}

ограничения типизирующих параметров_{опционально}

{ тело класса };_{опционально}

В С# символ «;» после закрывающей фигурной скобки тела класса не несёт какой-либо смысловой нагрузки. Он скорее удобен для С++ программистов, привыкших всегда ставить его.

Виды Членов Класса

Содержимое классов в C# можно разделить на 2 категории: **данные** и **функциональные члены**.

Данные:

- поля;
- константы.

Функциональные члены:

- методы;
- свойства;
- конструкторы;
- индексаторы;
- события;
- деструкторы;
- перегруженные операции.

Локальные Переменные и Константы

Синтаксис объявления локальной переменной:

// Локальные переменные метода можно объявить без инициализации,
// однако их нельзя использовать неинициализированными.
Тип имя_переменной [= выражение];

Синтаксис объявления локальной константы:

// Константы инициализируются сразу.
const Тип имя_константы = выражение;

Объявление Локальных Переменных: Пример

```
using System;
class Program
{
    static void Main()
    {
        DateTime time = DateTime.Now;
        Console.WriteLine(time);
        int j;
        // Console.WriteLine(j);
        Random rnd = new Random();
        j = rnd.Next();
        Console.WriteLine(j);
    }
}
```

Вывод:

19.09.2021 14:48:21

23479

// Без инициализации.

// CE – j не инициализирована.

// Инициализация j.

Контекстно-Ключевое Слово var

var позволяет не указывать тип локальной переменной — он выводится компилятором при инициализации.

Помните, что для полей использование var недопустимо.

Вывод:

19.09.2021 14:48:21
23479

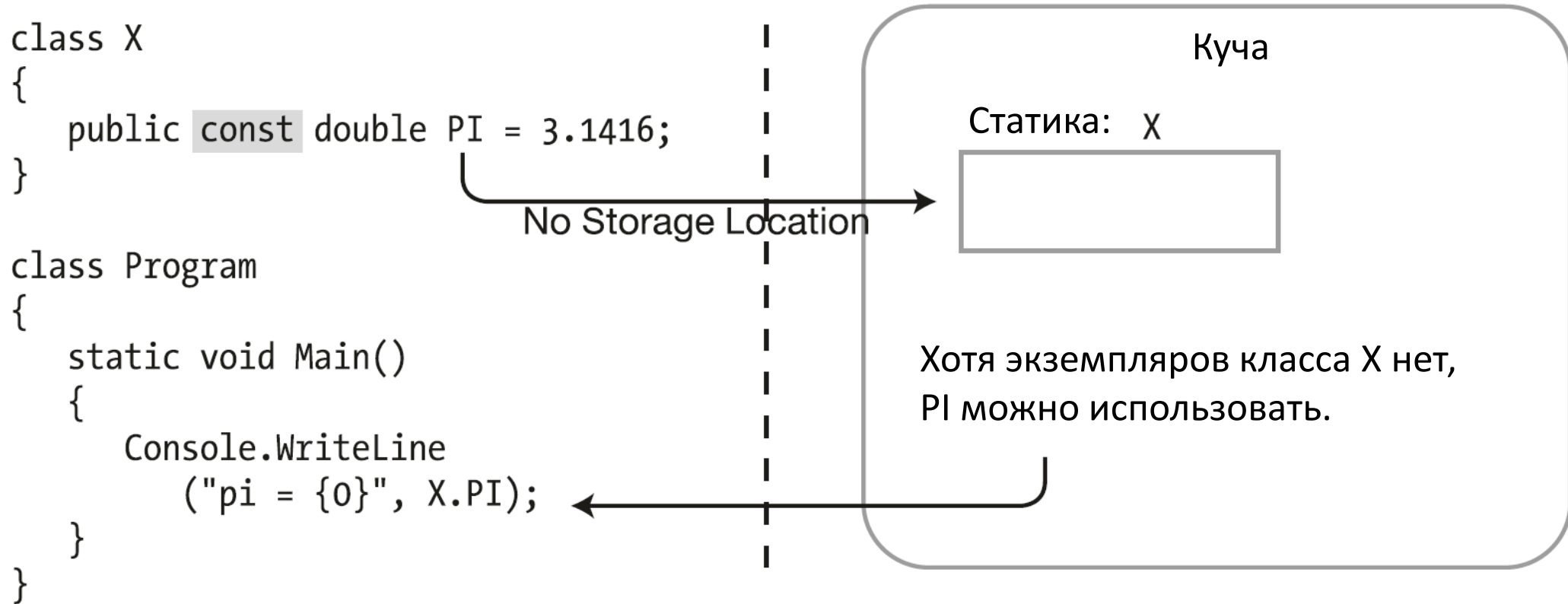
```
class Program
{
    static void Main()
    {
        var time = System.DateTime.Now;
        System.Console.WriteLine(time);
        var rnd = new System.Random();
        var j = rnd.Next();
        System.Console.WriteLine(j);
    }
}
```

Модификатор const в C#

```
class Program
{
    // Константы доступны по имени типа.
    const int ten = 10;
    static void Main()
    {
        const double PI = 3.1415; // локальная константа.
        System.Console.WriteLine($"PI * ten = {PI * ten}");
    }
}
```

Модификатор const используется в C# для констант уровня компиляции и требует обязательной инициализации в момент определения.

Константы не Хранятся в Памяти



Важно: константы в C# в принципе не хранятся в памяти – они подставляются в непосредственное место их использования компилятором.

Объявление Статических Полей Класса

```
class MyClass
{
    static double sum;           // sum равно 0.0.
    static int x = 2, z, g = 9;  // z равно 0.
    static bool flag;           // flag равно false.
    static string ss;           // ss равно null.
}
```

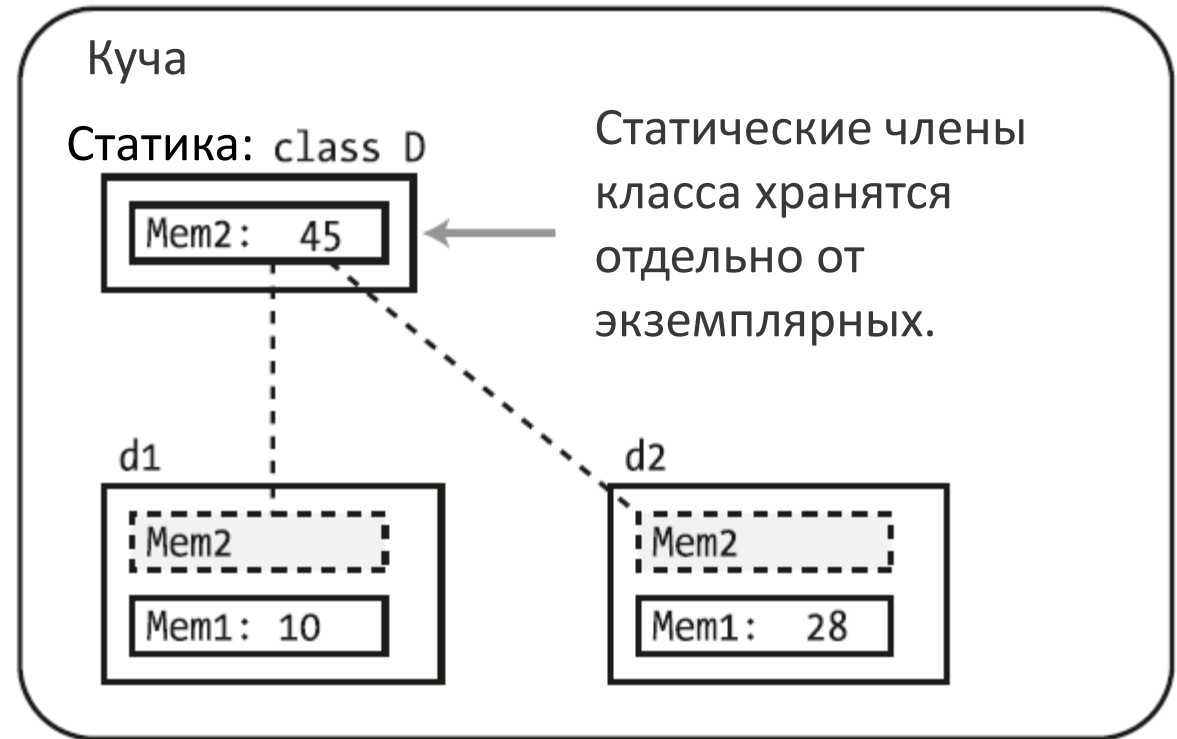
Помните: В отличие от локальных переменных, статические поля объявляются вне методов и **всегда** имеют значение по умолчанию.

Они получают эти значения в момент первого обращения к содержащему их типу перед вызовом статического конструктора.

Статические Поля vs. Поля Объектов

```
class D
{
    int Mem1;
    static int Mem2;
    ...
}

static void Main()
{
    D d1 = new D();
    D d2 = new D();
    ...
}
```



Статический Конструктор

```
using System;
class MyClass {
    // num инициализируется перед вызовом статического конструктора.
    static int num = 42;
    static MyClass() { // Статический конструктор: static <Имя Типа>.
        Console.WriteLine($"The answer is... {num}!");
        Console.WriteLine($"Starting time: {DateTime.Now}");
    }
}
```

Статический конструктор – специальный функциональный член, вызывающийся автоматически при первом обращении к типу сразу после инициализации всех статических полей. Это бывает полезным, когда необходимо организовать логирование или подгрузить какие-либо данные.

Статический Конструктор: Пример

```
using System;
class RandClass {
    private static Random rand;           // Private static field
    static RandClass() {                 // Static constructor
        rand = new Random();             // Initialize RandomKey
    }
    public static int GetRandomInt() => rand.Next();
}
class Program {
    static void Main() {
        Console.WriteLine($"Random #: {RandClass.GetRandomInt()}");
        Console.WriteLine($"Random #: {RandClass.GetRandomInt()}");
    }
}
```

Результат выполнения:

Random #: 580204170

Random #: 1415132876

Класс со Статическими Методами

```
using System;
class Program
{
    static long num = 13;
    static void Print(int u)
    {
        long prod = u * num;
        Console.WriteLine("u * num = " + prod);
    }
    static void Main()
    {
        Print(3);
    }
}
```

// Объявление класса Program.

// Заголовок метода Print.

// Конец метода Print.

// Заголовок метода Main.

// Конец метода Main.

// Конец класса Program.

Синтаксис объявления статического метода:

static <тип возвращаемого значения> <Имя>([Параметры...]) { <Тело> }

Модификаторы Доступа

Модификаторы доступа – определяют, можно ли обращаться к определённому типу/члену типа за его пределами.

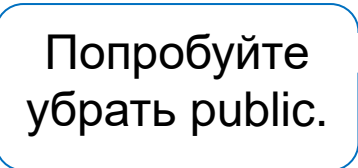
Всего существует 6 модификаторов доступа:

- **private** – закрытый (доступ только внутри типа);
- **public** – открытый (доступ без ограничений);
- **protected** – защищенный (доступ только для наследников);
- **internal** – внутренний (доступ внутри той же сборки);
- **protected internal** – внутри сборки ИЛИ для всех наследников;
- **private protected** – только для наследников внутри той же сборки (C# 7.2).

Важно: по умолчанию члены пространств имён *неявно* имеют модификатор **internal**, а все члены классов – **private**.

Доступность Членов Вне Класа

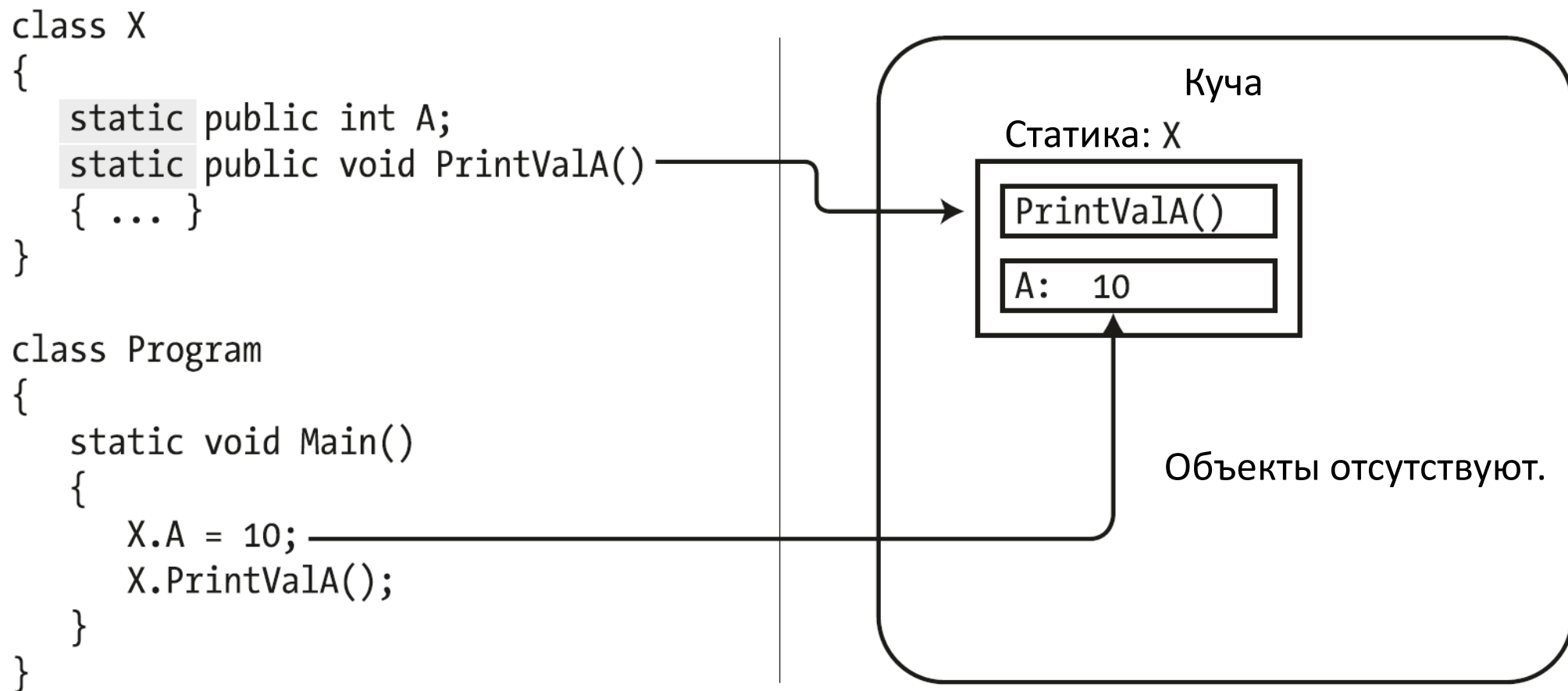
```
class MyClass {  
    static long num = 13;  
    public static void Print(int u) {  
        long prod = u * num;  
        System.Console.WriteLine("u * numb = " + prod);  
    }  
}  
  
class Program {  
    static void Main() {  
        MyClass.Print(3);    // Обращение к Print по имени типа.  
    }  
}
```



Попробуйте
убрать public.

Если убрать модификатор *public* у метода `Print`, то он будет *неявно* иметь *приватный* (*private*) доступ (доступ только внутри типа `MyClass`).

Доступ к Статическим Членам Класса



Помните, что при обращении к статическим членам некоторого типа объекты данного типа могут отсутствовать.

Добавление Файлов с Исходным Кодом на C# в Visual Studio 2019

На практике размещать все методы и классы в одном файле оказывается неудобно, а в случае изменений приходится перекомпилировать весь код.

Чтобы создать отдельный класс в Visual Studio 2019 нужно выполнить несколько шагов:

- 1) Открыть обозреватель решений (Solution Explorer, *Ctrl+Alt+L*);
- 2) Нажать правой кнопкой мыши по имени проекта (не решения!);
- 3) Выбрать пункт «Добавить» (Add) в выпадающем меню;
- 4) Выбрать «Класс...» (Class...) во вложенном меню;
- 5) Ввести в появившемся поле имя класса и нажать кнопку «Добавить» (Add).

Проект с Двумя Файлами Кода

```
// Файл Separate.cs
public class Separate {
    public static double Average(double x, double y) {
        return (x + y) / 2;
    }
}
```

Без public
возникнет
ошибка
доступа.

```
using System;
// Файл Program.cs
class Program {
    static void Main() {
        double average = Separate.Average(3, 6);
        Console.WriteLine($"Average = {average}");
    } //end of Main
} // end of Program
```

Операции Объединения с null

Операция	Эффект
$x \ ?? \ y$	Возвращает значение x , если он не null или вычисляет значение y и возвращает его.
$x \ ??= \ y$	Вычисляет значение y и присваивает его x только в случае, когда x оказался равен null.

Операции объединения с null предоставляют лаконичный синтаксис для выполнения вычислений в случае, когда левый операнд оказывается равен null.

Это может быть крайне удобно в случаях, когда нужна отложенная инициализация.

Операция Объединения с null: Пример

```
using System;
class Program {
    static Random rnd; // Статическое поле по умолчанию равно null.
    static int SumRandom(int left, int right) {
        // Создаём объект Random при первом использовании.
        rnd ??= new Random();
        return rnd.Next(left, right) + rnd.Next(left, right);
    }
    static void Main() {
        int left = int.Parse(Console.ReadLine());
        int right = int.Parse(Console.ReadLine());
        Console.WriteLine($"Random Sum: {SumRandom(left, right)}");
    }
}
```

Разделяемые типы (partial types)

```
public partial class Employee
{
    public void DoWork()
    {
    }
}
```

```
public partial class Employee
{
    public void GoToLunch()
    {
    }
}
```

ЭКВИВАЛЕНТНО

```
public class Employee
{
    public void DoWork()
    {
    }

    public void GoToLunch()
    {
    }
}
```