

**Лекция 9.**  
**Массивы.**  
**Отладка.**  
**Аргументы программы**

# Задача

- Хотим прочитать с консоли 2 числа, чтобы потом работать с ними
- Как это сделать?
- А что если хотим прочитать 5 чисел?
- А если N чисел, где N вводят с консоли (то есть оно заранее неизвестно)?
- Чтобы решать такие задачи, есть специальная структура данных **массив**, которая позволяет в одной переменной хранить много однотипных значений

# Массивы

- **Массив** – это тип данных, который хранит в себе фиксированное количество элементов одного типа
- Массив объявляется при помощи квадратных скобок:

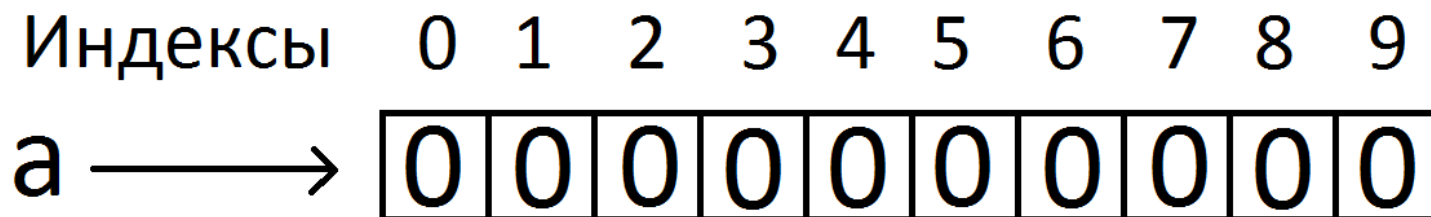
```
Тип[] имя = new Тип[размер];
```

- **Пример:**

```
int[] a = new int[10]; // массив из 10 целых чисел
```

# Массивы

- Массивы являются объектами (ссылочными типами)
- Массивы хранятся в памяти единым куском
- **Индексы** (номера) элементов массива отсчитываются от нуля
- `int[] a = new int[10];` // массив из 10 int



# Массивы

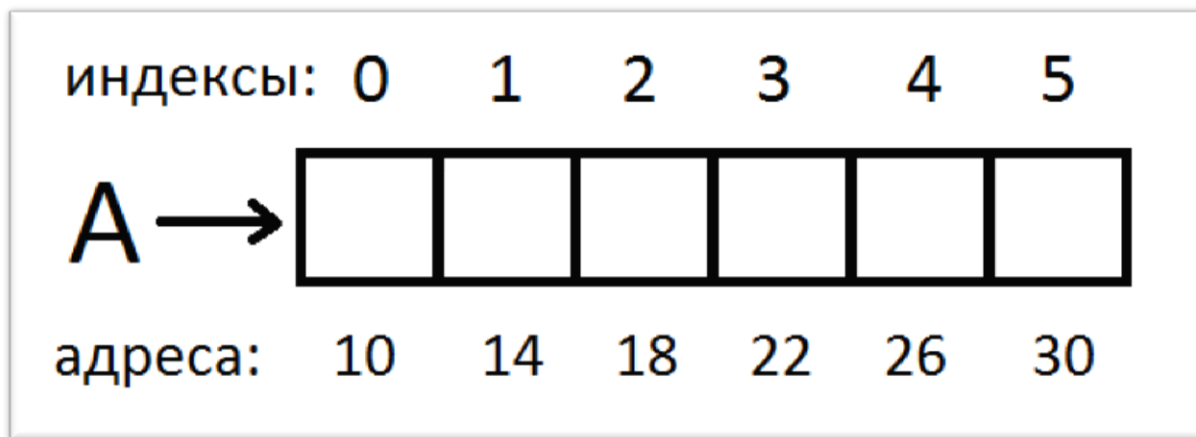
- `int[] array = new int[10];`
- При объявлении массива, все его элементы инициализируются значениями по умолчанию: 0 для числовых типов, `false` для `bool`, `null` – для ссылочных типов
- Массивы являются объектами и передаются в функции по ссылке

# Обращение к элементам массива

- Обращаться к элементам массива можно при помощи квадратных скобок:
- ```
int[] array = new int[10];  
array[0] = 1;  
Console.WriteLine(array[0]);    // 1  
array[10] = 4; // ошибка времени исполнения, выход за  
границы массива
```
- Потому что индексы отсчитываются от 0
- Т.е. в массиве размером N будут элементы с номерами от 0 до N-1 включительно

# Доступ к массиву по индексу

- Массив знает, с какого адреса в памяти он начинается и каков размер типа данных, элементы которого он содержит
- Тогда, зная это, можно легко вычислить адрес элемента по его индексу
- Пусть, например, адрес начала массива  $A$  равен 10, массив хранит `int`, значит размер элемента – 4 байта
- Тогда адрес элемента  $A[5]$  равен  $10 + 5 * 4 = 30$
- Поэтому в массиве очень быстрый доступ по индексу



# Итерирование по массиву

- Часто бывает нужно пройти по всем элементам массива. Каждый массив имеет свойство `Length`, хранящее его длину
- Проход по массиву с печатью элементов:
- `// где-то выше объявлен массив array`  
`for (int i = 0; i < array.Length; ++i)`  
`{`  
 `Console.WriteLine(array[i]);`  
`}`



# Заполнение массива

- При присваивании элемента массива внутри [] можно указывать не только числа, но и любые выражения, которые выдают целое число
- Заполнение массива числами от 0 до 30:
- ```
int[] array = new int[31];  
for (int i = 0; i < array.Length; ++i)  
{  
    array[i] = i;  
}
```
- 0 1 2 3 4 ... 30

# Задача

- Написать программу, заполняющую массив длины 100 последовательными числами от 1 до 100
- После этого отдельным циклом распечатать элементы массива

# Итерирование по массиву

- Проход по массиву с печатью элементов:

- ```
for (int i = 0; i < a.Length; ++i)
{
    Console.WriteLine(a[i]);
}
```

- Цикл `foreach`:

- ```
foreach (int e in array)
{
    Console.WriteLine(e);
    // e – текущий элемент массива
}
```

# Невозможность изменения через foreach

- Цикл `foreach` не позволяет изменять элементы массива
- `foreach (int e in array)`  
{  
 `e = 3;`  
 `// поменялась переменная e, а не элемент`  
 `// массива`  
}

# Итерирование по массиву

- Если нужно пройти по всему массиву, не важен индекс и не нужно изменять элементы, то следует применять `foreach`
- Иначе - следует применять циклы `for`, `while`, `do-while`

# Зачем нужен foreach

- Цикл `foreach` проще, чем привычные циклы вроде `for`
- ```
for (int i = 0; i < a.Length; ++i)
{
    Console.WriteLine(a[i]);
}
```
- ```
foreach (int e in array)
{
    Console.WriteLine(e);
}
```
- Исходя из ограничений `foreach` уже сразу видно, что идет проход по всем элементам, в прямом порядке, и что массив при этом не меняется
- Это важно для простоты читаемости кода

# Задача

- В задаче про заполнение массива замените второй цикл на цикл `foreach`

# Инициализация массива

- Краткое объявление массива:
- `int[] a = {1, 2, 3, 4, 5, 6, 7};`  
`// длина вычислится сама`
- Есть еще такой вариант:
- `int[] a = new int[] {1, 2, 3, 4, 5, 6, 7};`  
`// этот вариант можно использовать в return`
- Массивы могут быть любого типа. Например:
- `string[] s = {"Pavel", "Artem"};` `// массив строк`
- `int[][] a = new int[10][];` `// массив массивов`



# Многомерные массивы

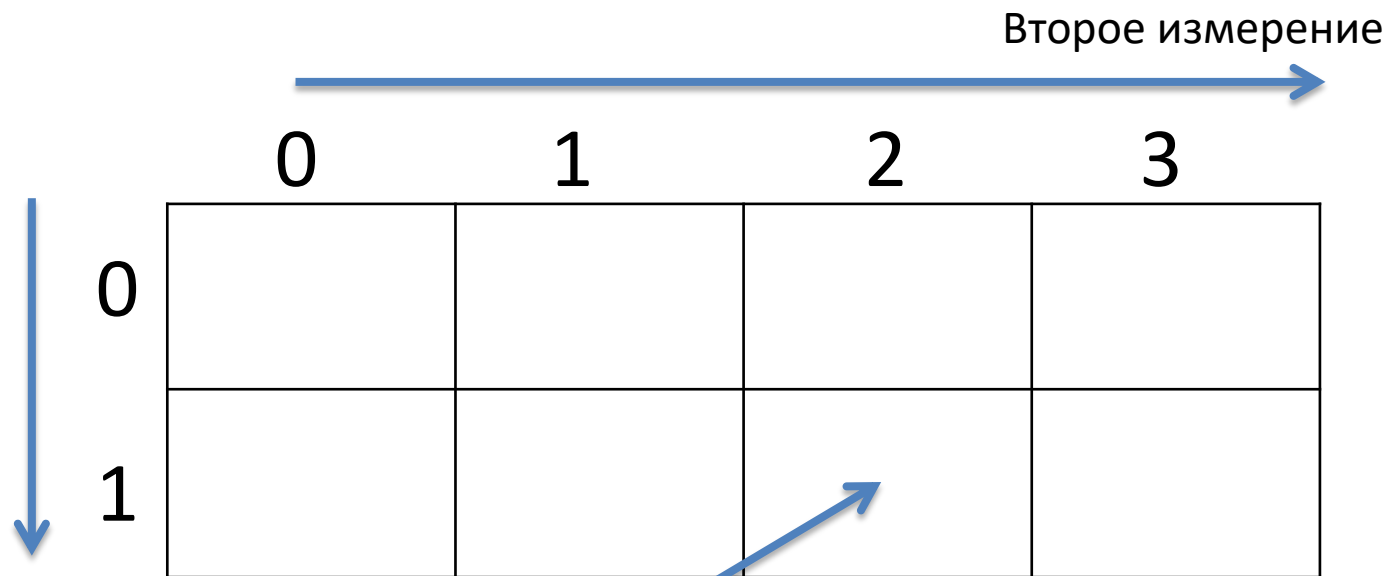
- Это массивы, элементами которых являются массивы
- В C# есть два варианта многомерных массивов:
  - У которых размеры измерений фиксированы
  - У которых размеры измерений могут быть разными

# Фиксированные многомерные массивы

- `int[,] a = new int[3, 5];`
- Это массивы, элементами которых являются массивы
- Обратиться к конкретному элементу можно, указав индексы для каждого **измерения** массива:
- `int x = a[1, 4];`

# Фиксированные многомерные массивы

- Двумерный массив можно представить себе как таблицу
- `int[,] a = new int[2, 4];`



- `int x = a[1, 2];`

# Проход по многомерному массиву

- `int[,] a = new int[3, 5];`
- Свойство `Length` для такого массива выдаст общее количество элементов (будет 15)
- А что если хотим пройти по строкам и столбцам в двух вложенных циклах?
- ```
for (int i = 0; i < a.GetLength(0); ++i)
{
    for (int j = 0; j < a.GetLength(1); ++j)
    {
        Console.WriteLine(a[i, j]);
    }
}
```

`GetLength(x)` выдает размер измерения с номером `x`

# Фиксированные многомерные массивы

- Измерений может быть сколько угодно
- `int[,,,] a = new int[3, 5, 2];`
- Количество запятых в названии типа – это количество измерений массива + 1
- В правой части указываются размерности соответствующих измерений

# Краткий синтаксис

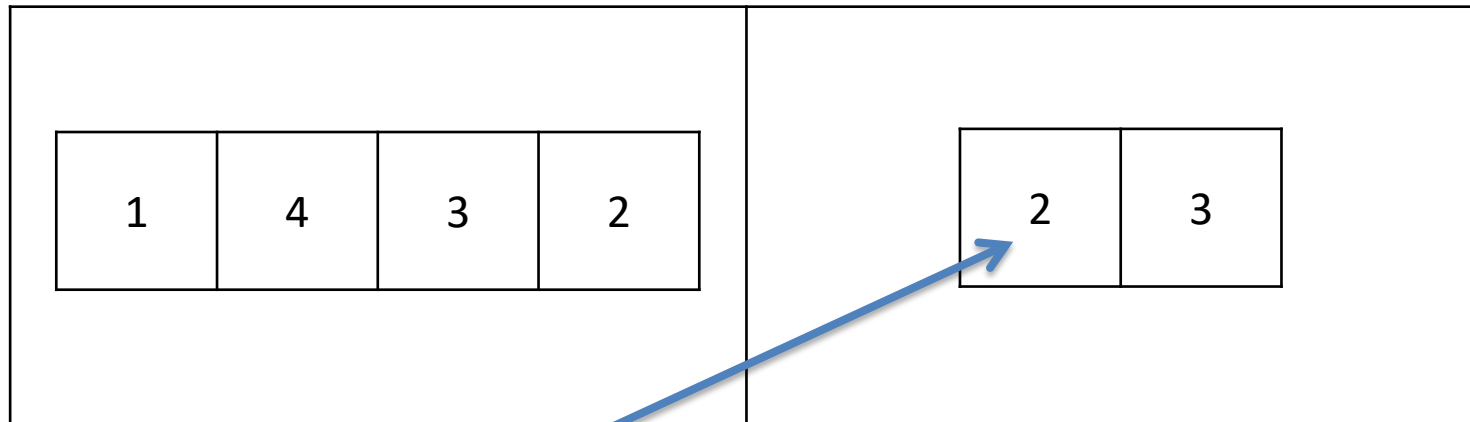
- Массив размера 3, 2
- `int[,,]` `a =`  
    {  
        { 2, 3, 2 },  
        { 4, 10, 1 }  
    };

# Нефиксированные многомерные массивы

- `int[][] a = new int[3][];`
- Это тоже массивы, элементами которых являются массивы
- Длину можно указать только у первого измерения
- Создается массив с таким количеством элементов, забитый `null`'ами
- Каждый элемент – это массив, который можно создать самим, причем любой длины
- То есть тут уже размеры измерений могут отличаться

# Многомерные массивы

- Такой массив удобнее представлять как вложенные массивы
- `int[][] a = new int[2][];`  
`a[0] = new int[] { 1, 4, 3, 2 };`  
`a[1] = new int[] { 2, 3 };`



- `int x = a[1][0];`  
`int[] y = a[0];` // получили весь массив



# Стандартные функции

- Частые операции с массивами уже реализованы в C#: копирование, сортировка, поиск, печать массива и т.д.
- Для них есть стандартные функции, например, в классе `Array`
- Поэтому при решении задач на практике привыкайте искать – скорее всего, кто-то уже сделал это за вас

# Аргументы программы

# Аргументы программы

- Объявление функции Main:
- `public static void Main(string[] args)`  
`{`  
`}`
- Т.е. функция Main принимает массив строк-параметров
- Это позволяет запускать программу с заданными параметрами
- Например, программе можно указать путь к файлу, с которым она должна работать

# Как использовать параметры?

- Есть 2 частых варианта использования:
  - Просто использовать параметры в качестве входных данных
    - Например, это могут быть пути к файлам
  - Выполнять разный код в зависимости от параметров
    - Например, это флаги, управляющие режимами программы
    - Смотрим, какой параметр передал пользователь, сравниваем с известными нам значениями /all и т.д., и в зависимости от этого выполняем разный код
- Демонстрация обоих способов

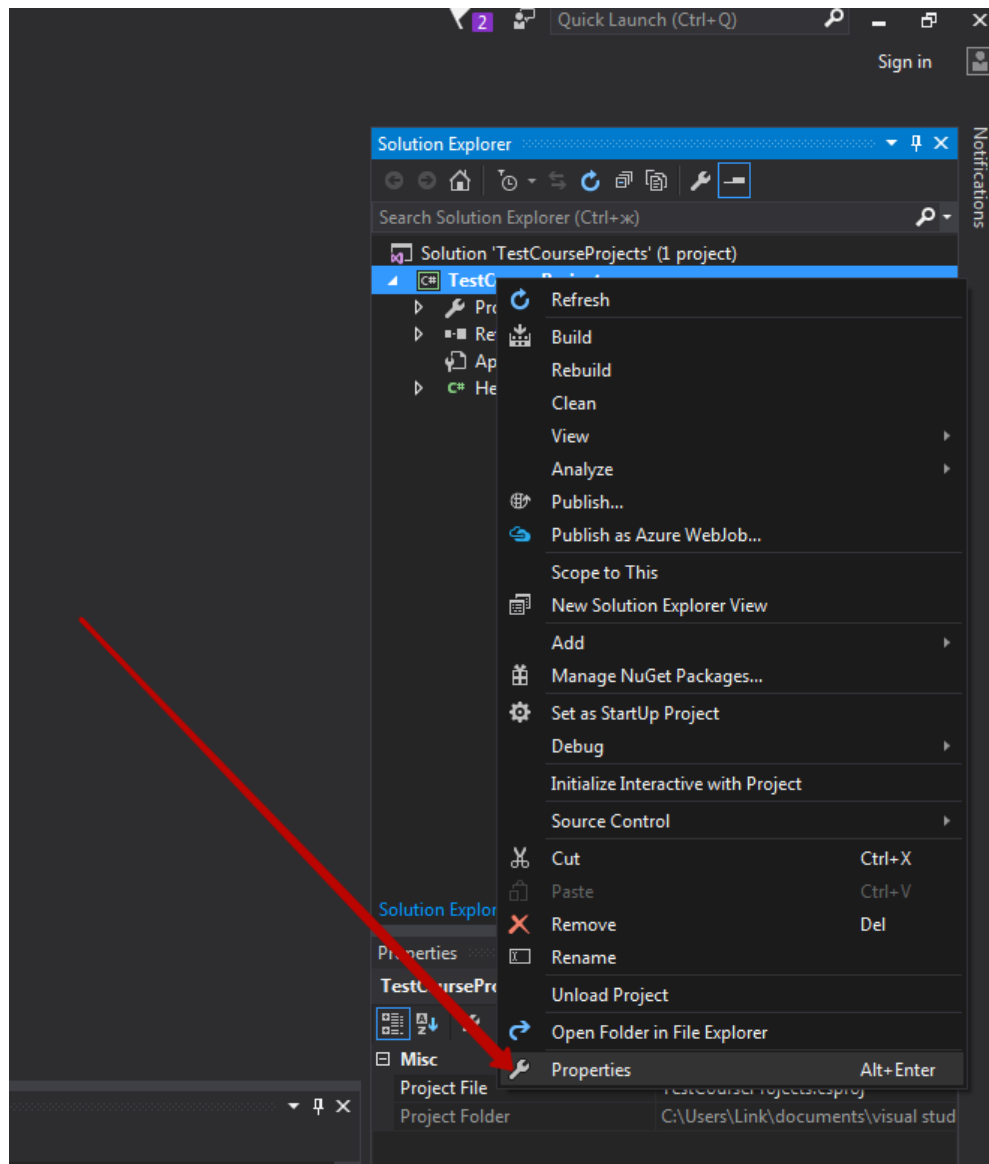
# Программы с параметрами

- Если программа принимает параметры командной строки, то нужно:
  - Сделать отдельную команду `–help`, которая будет выводить справку, чтобы пользователи могли узнать, как этой командой пользоваться
  - Проверять количество переданных аргументов. Если оно некорректное, то надо выдать об этом сообщение, и вывалить `help`
  - Проверять сами аргументы на корректность. Если что, выдавать сообщение

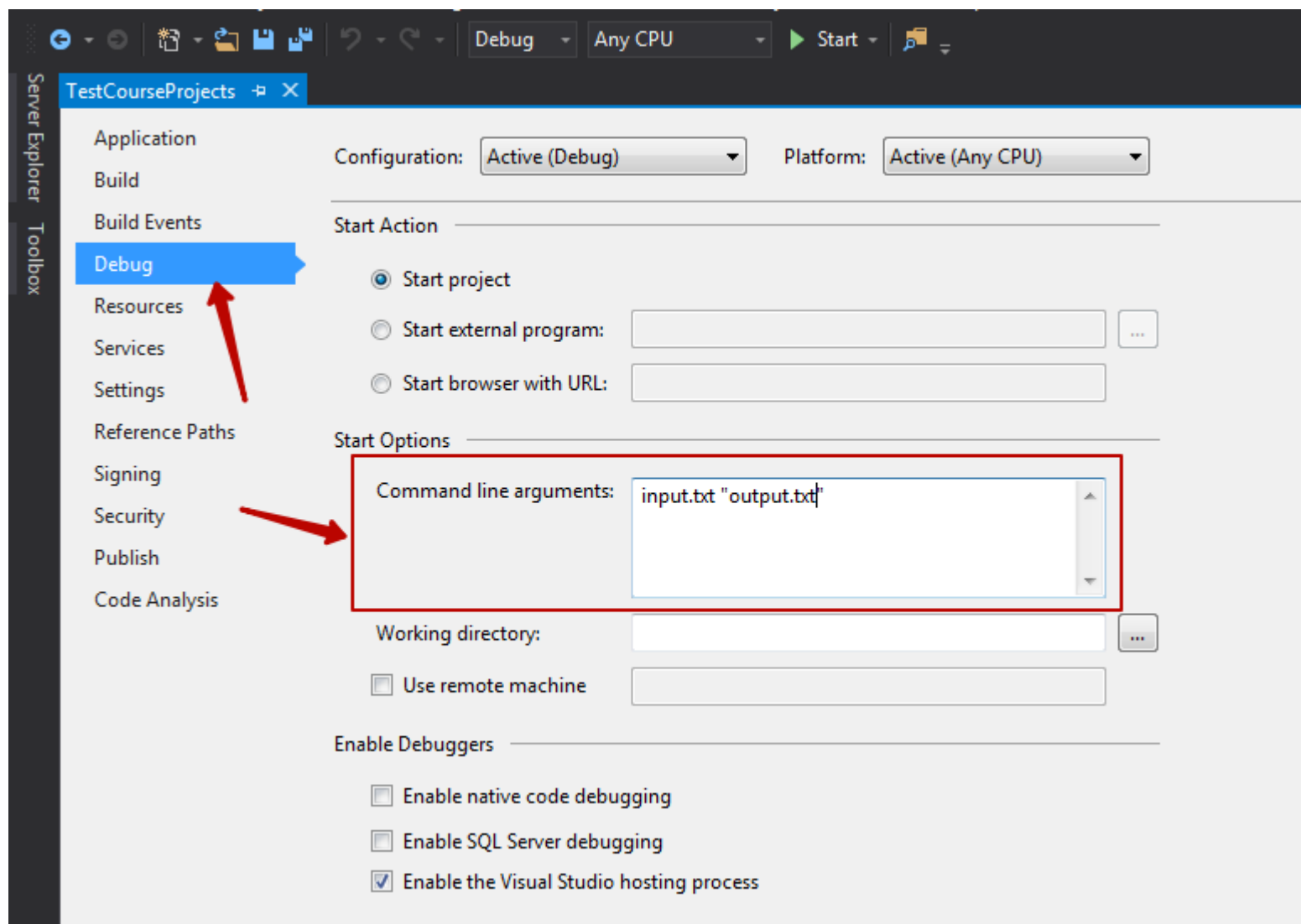
# Отладка программ с аргументами

- Для удобства среды разработки позволяют запустить вашу программу с нужными аргументами командной строки
- Демонстрация в IDEA и Visual Studio

# Аргументы программы



# Аргументы программы





# Аргументы программы

- Параметры разделяются пробелами
- Если в значении параметра есть пробел, то значение нужно заключать в двойные кавычки
- Примеры:
  1. `input.txt output file.txt`  
// 3 параметра – `input.txt`, `output`, `file.txt`
  2. `input.txt "output file.txt"`  
// 2 параметра – `input.txt` и `output file.txt`

# Задача

- Передать программе параметры
- Вывести в консоль количество параметров
- Вывести в консоль значения параметров при помощи цикла `foreach`

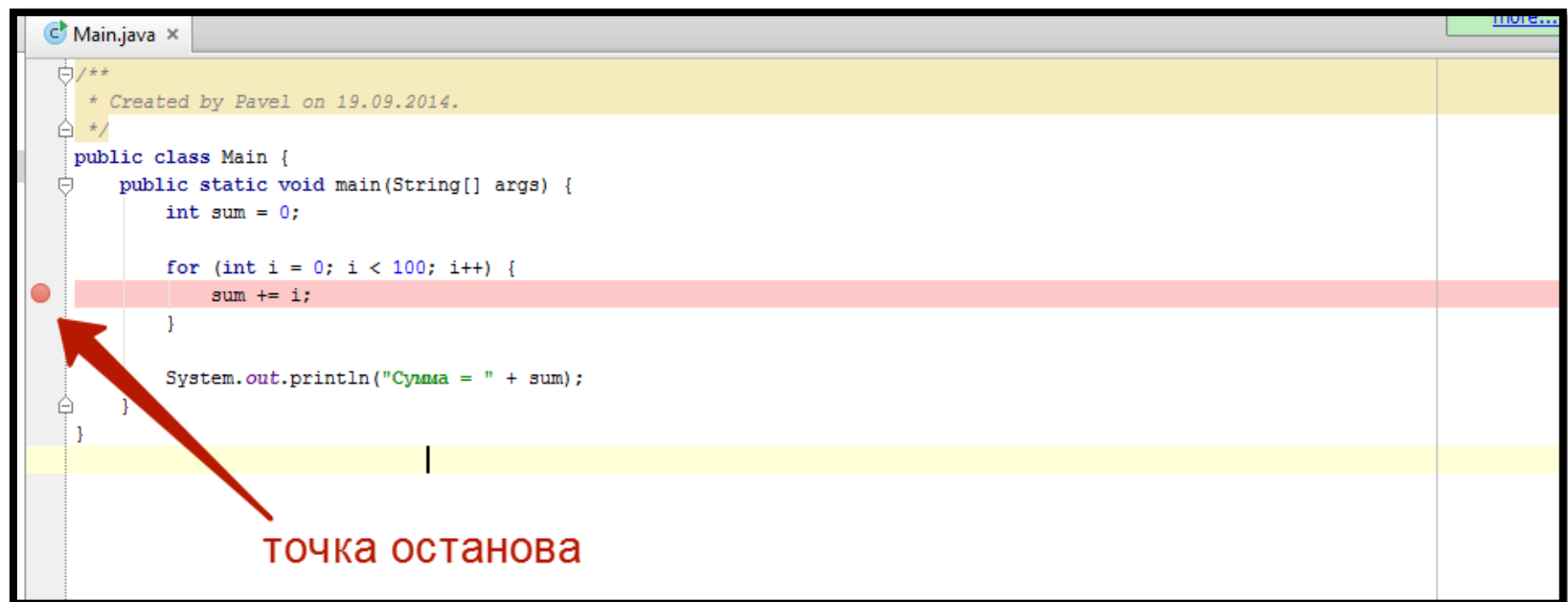
**Отладка**

# Отладка программ

- **Отладка программ** – процесс поиска ошибок
- По-английски – **debug**
- Среды разработки, в том числе Visual Studio предоставляют удобные средства отладки

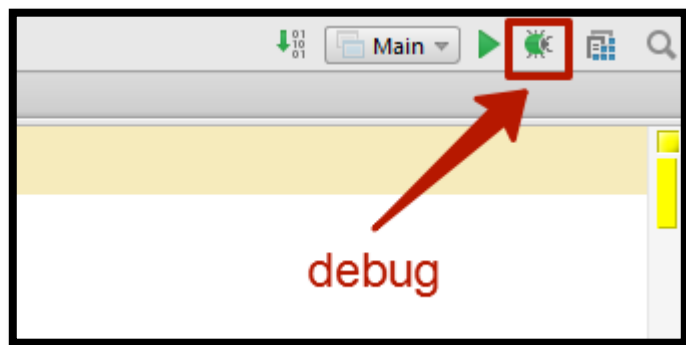
# Точки останова

- Точки останова (**breakpoints**)
- Позволяют остановить исполнение программы в указанном месте, когда поток исполнения достигнет его
- Добавляются/убираются кликом по столбцу слева



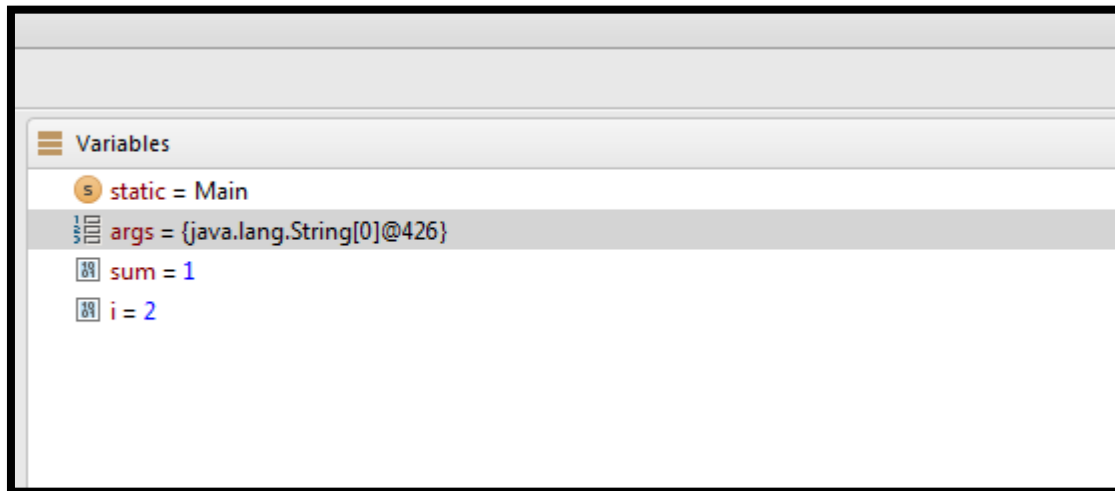
# Запуск отладки

- Если запускать программу через Run, то исполнение не останавливается на точках останова
- Для отладки нужно запускать программу через Debug



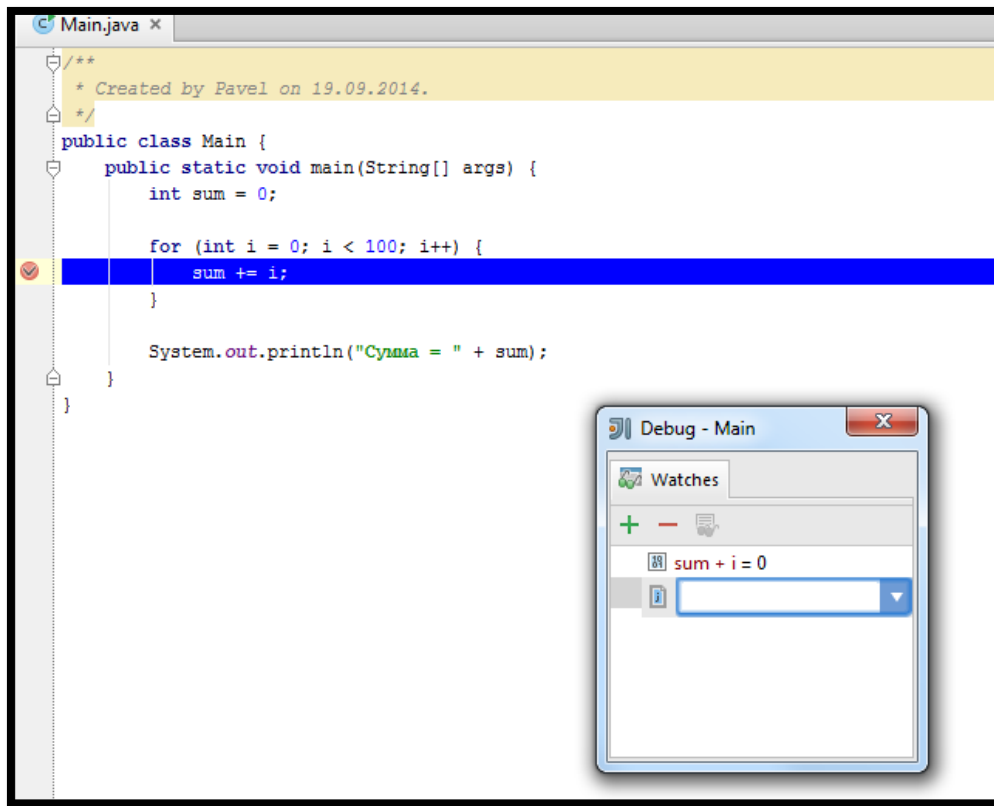
# Просмотр значений переменных

- Когда программа остановлена, можно посмотреть текущие значения переменных



# Просмотр результатов выражений

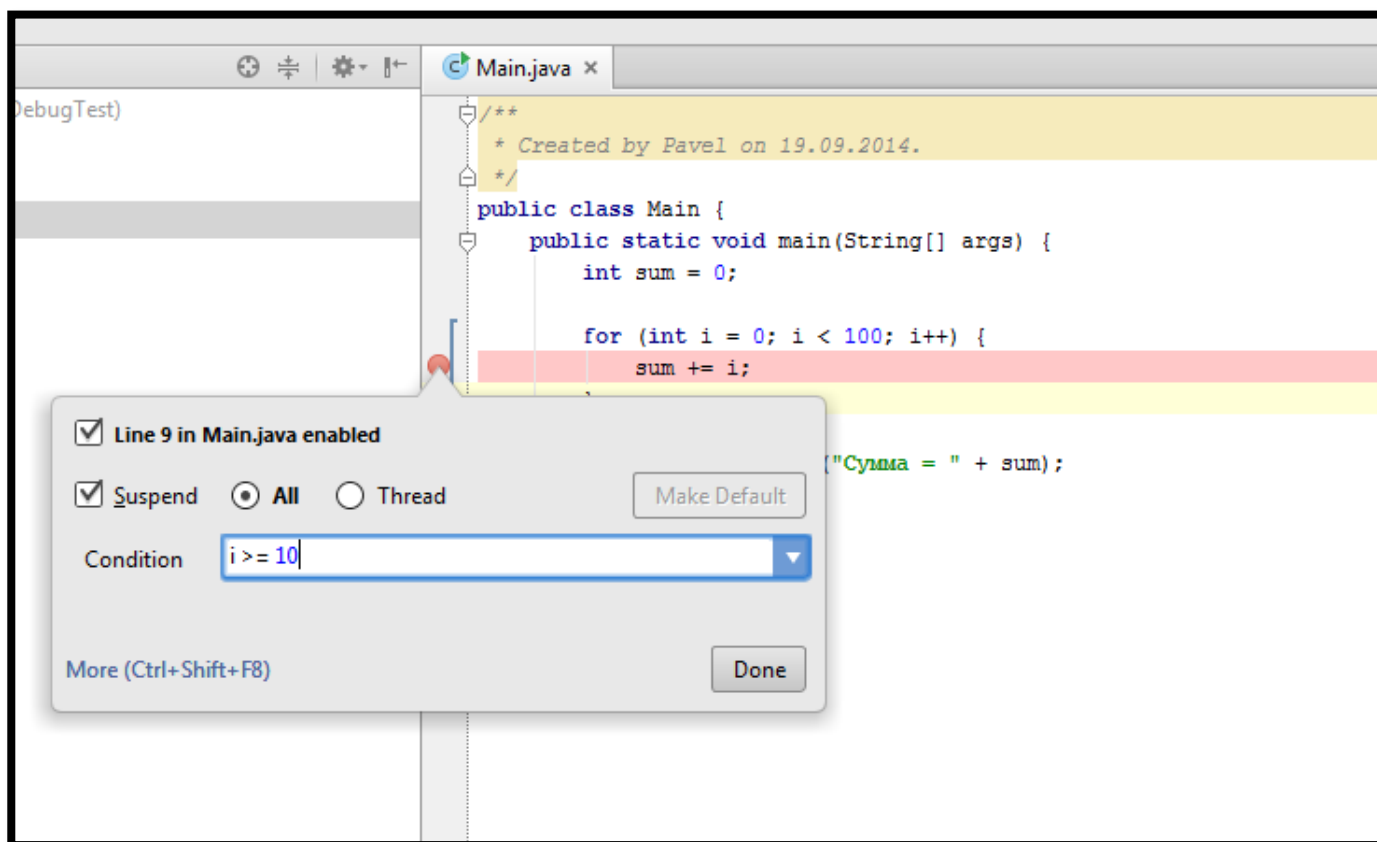
- Когда программа остановлена, можно посмотреть значение любого выражения, которое хочется проверить





# Точки останова с условием

- Для точки останова можно задать условие когда она будет срабатывать

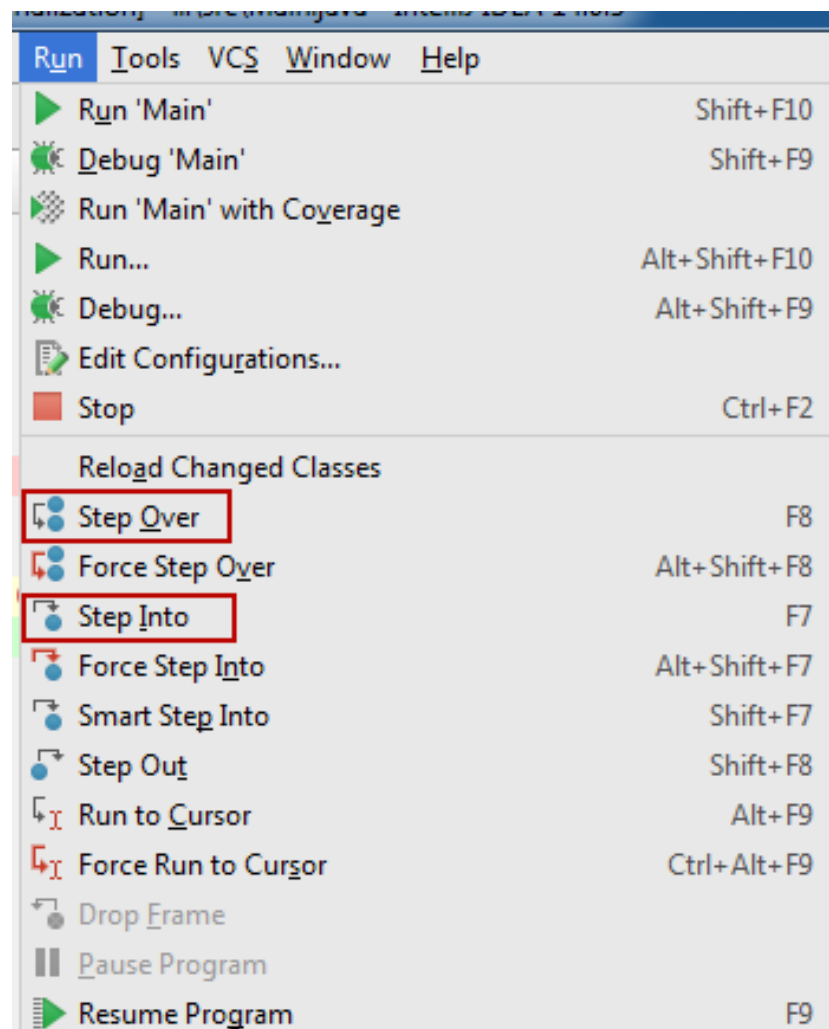


# Пошаговая отладка

- Часто бывает полезна **пошаговая отладка** – по нажатию кнопки будет выполняться по одной команде
- Есть два вида пошаговой отладки:
  - с заходом в функцию (**step into**) – F7
  - без захода в функцию (**step over**) – F8

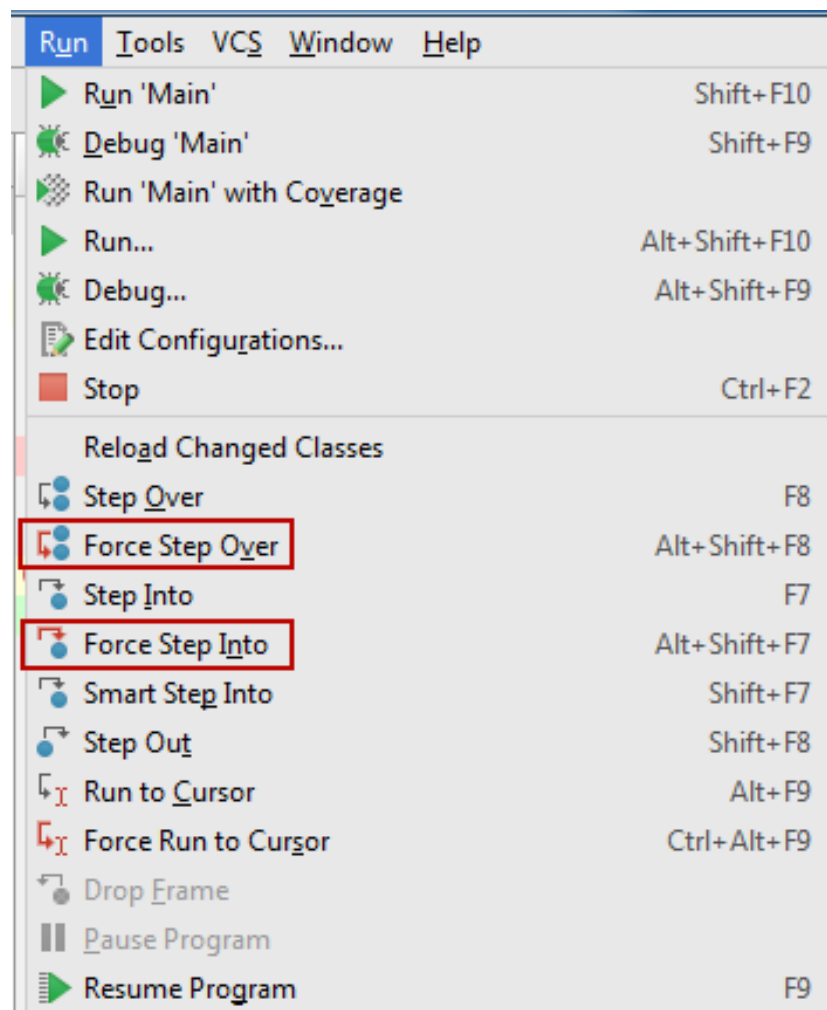
# Пошаговая отладка

- С заходом в функцию (**Step Into**) – F7
- Без захода в функцию (**Step Over**) – F8



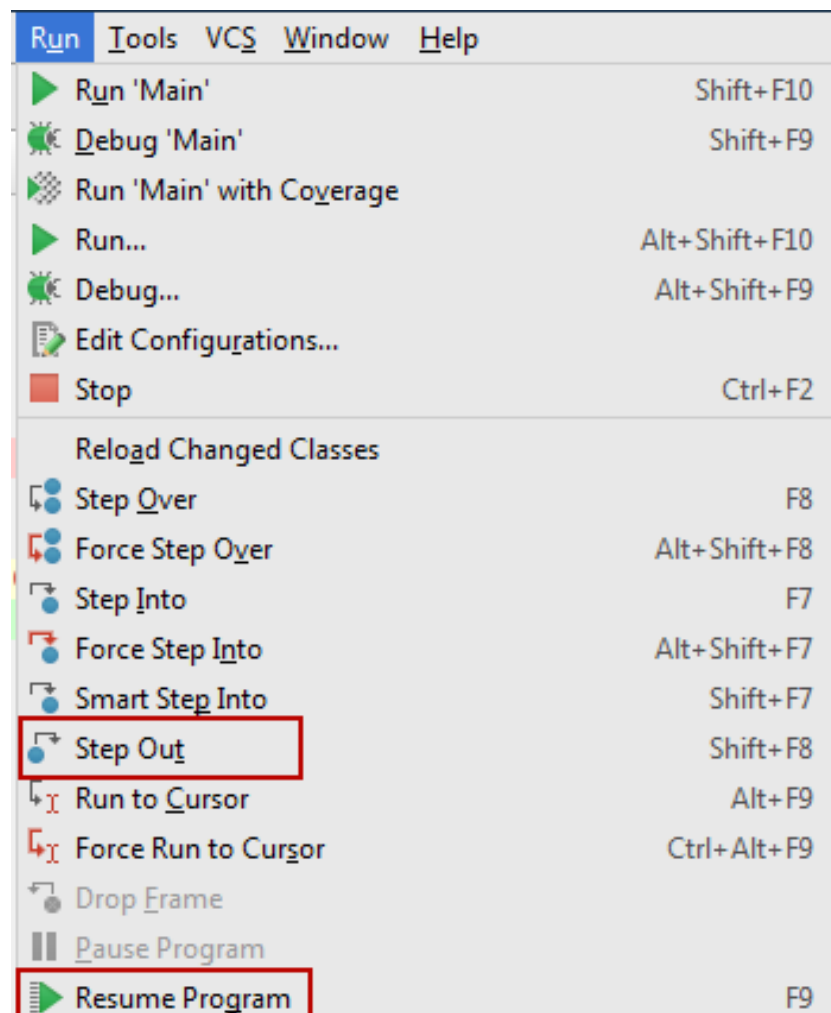
# Пошаговая отладка

- Принудительный заход в функцию (**Force Step Into**)  
— позволяет зайти в исходный код библиотеки Java
- Принудительный пропуск функции (**Force Step Over**)  
— пропускает функцию, не останавливаясь на точках останова внутри нее



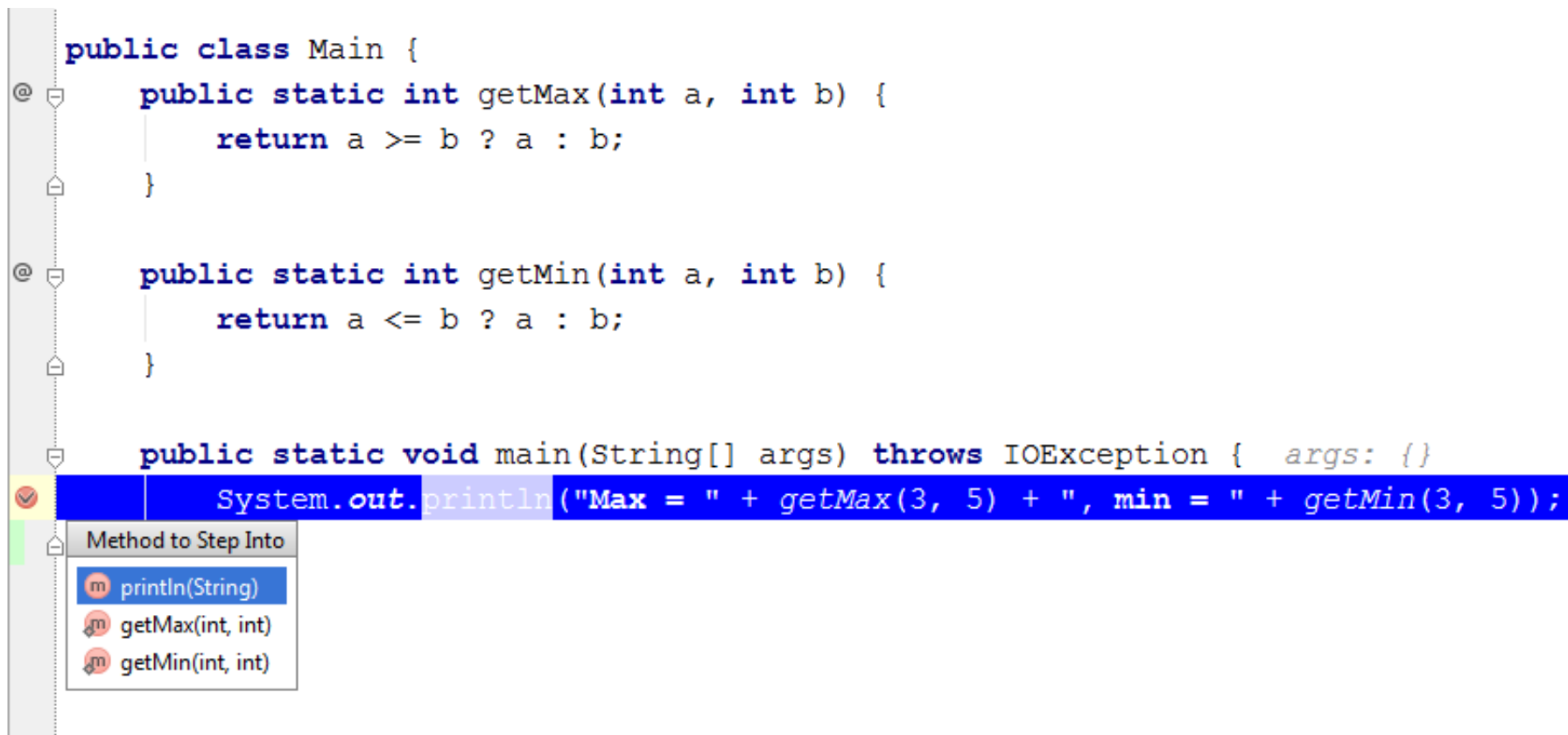
# Пошаговая отладка

- Кнопка **resume** позволяет продолжить исполнение до следующей точки останова
- Команда **Step Out** - переход к следующей команде, которая будет исполняться после окончания метода



# Пошаговая отладка

- **Smart Step Into (Shift + F7)** – если в одной строке несколько вызовов функций, позволяет выбрать куда переходить



```
public class Main {  
    public static int getMax(int a, int b) {  
        return a >= b ? a : b;  
    }  
  
    public static int getMin(int a, int b) {  
        return a <= b ? a : b;  
    }  
  
    public static void main(String[] args) throws IOException { args: {}  
        System.out.println("Max = " + getMax(3, 5) + ", min = " + getMin(3, 5));  
    }  
}
```

The screenshot shows a Java code editor with a vertical toolbar on the left. The toolbar includes icons for breakpoints, a 'Smart Step Into' icon (a red circle with a white 'm'), and a 'Step Into' icon (a green circle with a white 'm'). The 'Smart Step Into' icon is highlighted, and a context menu is open below it. The menu is titled 'Method to Step Into' and lists three methods: 'println(String)', 'getMax(int, int)', and 'getMin(int, int)'. The 'println(String)' method is selected, indicated by a blue highlight. The code being debugged is a Java class named 'Main' with two static methods, 'getMax' and 'getMin', and a 'main' method. The 'main' method calls 'System.out.println' with two arguments, 'getMax(3, 5)' and 'getMin(3, 5)'. The line containing the 'println' call is highlighted in blue.

# Задача

- Попрактикуйтесь в отладке какой-нибудь своей программы
- Попробуйте точки останова, точки останова с условием, посмотрите значения переменных, результаты выражений
- Попробуйте инструменты пошаговой отладки

# Задача на дом «Поиск максимума»

- Написать функцию, которая ищет максимальное число в массиве вещественных чисел



# Задача на дом «Поиск элемента»

- Написать функцию, которая ищет указанное число в массиве, и если находит его, то выдает его индекс. А если не находит, то выдает -1

# Задача на дом «Массив строк в верх. рег»

- Написать функцию, которая принимает массив строк и изменяет его, присваивая элементам эти же строки, но в которых все символы заглавные. Для этого использовать метод класса `String` `ToUpper()`
- Пример:  
`string s = "hello";`  
`string b = s.ToUpper(); // "HELLO"`

# Задача на дом «Среднее арифм. массива»

- Найти среднее арифметическое элементов массива, которые являются четными числами

# Задача на дом «Разворот массива»

- Переставить элементы массива в обратном порядке

# Задача на дом «Проверка сортировки»

- Написать функцию, которая проверяет, что массив отсортирован по возрастанию
- И написать функцию, которая проверяет, что массив отсортирован по убыванию

# Задача на дом «Таблица умнож массив»

- Написать функцию, которая создает двумерный массив с таблицей умножения
- Размер таблицы должен быть параметром функции
- Вызвать функцию и распечатать результат в Main