

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

Задача

- Хотим прочитать с консоли 2 числа и найти их сумму
- 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:

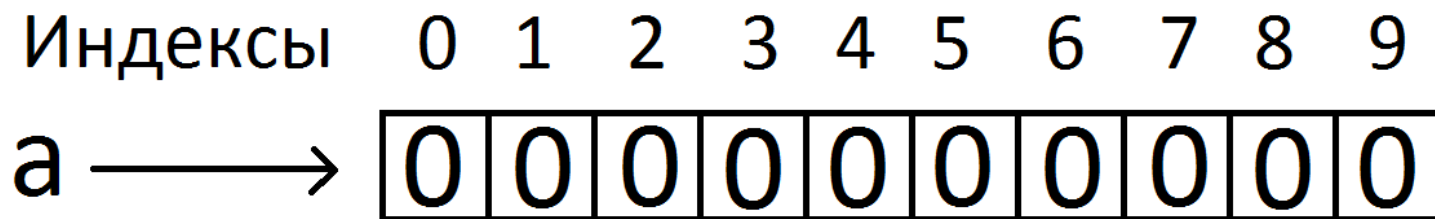
```
Тип[] имя = new Тип[размер];
```

- **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 `boolean`, `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;`
`System.out.println(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

Итерирование по массиву

- Часто бывает нужно пройти по всем элементам массива. Каждый массив имеет поле `length`, хранящее его длину
- Проход по массиву с печатью элементов:
- `// где-то выше объявлен массив array`
`for (int i = 0; i < array.length; ++i) {`
 `System.out.println(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) {  
    System.out.println(a[i]);  
}
```
- Специальная версия цикла **for (foreach)**:
- ```
for (int e: array) {
 System.out.println(e);
 // e – текущий элемент массива
}
```

# Невозможность изменения через foreach

- Цикл foreach не позволяет изменять элементы массива
- ```
for (int e: 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) {
 System.out.println(a[i]);
}
```
- ```
for (int e: array) {  
    System.out.println(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`
- Массивы могут быть любого типа. Например:
- `String[] s = {"Pavel", "Artem"};` `// массив строк`
- `int[][] a = new int[10][];` `// массив массивов`

Синтаксис объявления массива

- Более привычно объявлять массив так:

```
int[] a = new int[10];  
int[][] b = new int[3][5];
```

- Но можно и так:

```
int a[] = new int[10];  
int b[][] = new int[3][5];
```

- Эти варианты эквивалентны с точки зрения кода

Многомерные массивы

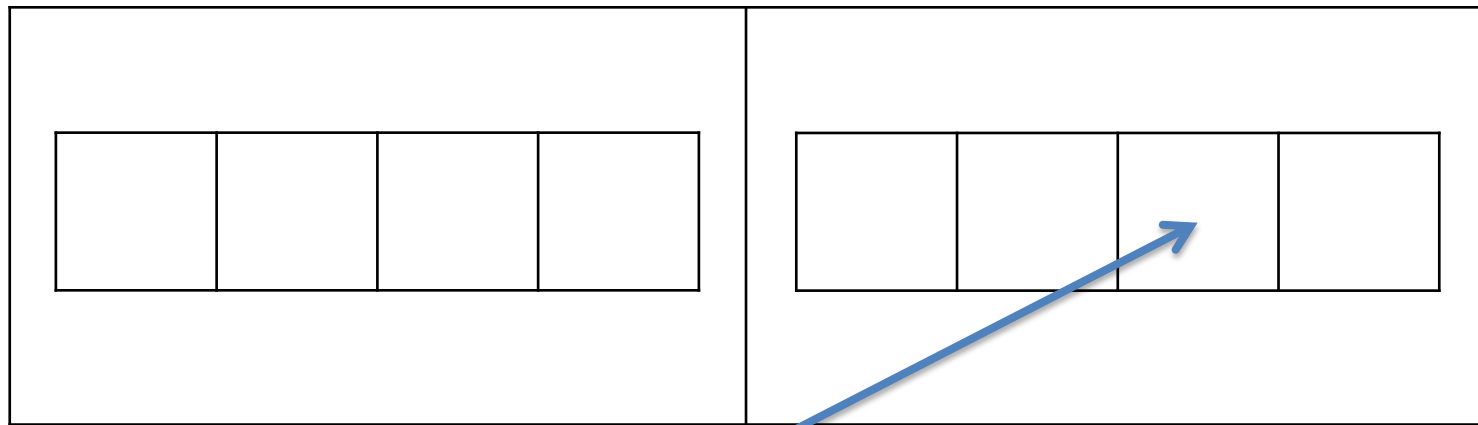
- `int[][] a = new int[3][5];`
- Это массивы, элементами которых являются массивы
- Обратиться к конкретному элементу можно, указав индексы для каждого **измерения** массива:
- `int x = a[1][4];`
`int[] y = a[1];` *// это массив*

Многомерные массивы

- Многомерный массив можно представить себе двумя способами:
 - Массив вложенных массивов
 - Многомерная таблица

Многомерные массивы

- Как вложенные массивы
- `int[][] a = new int[2][4];`

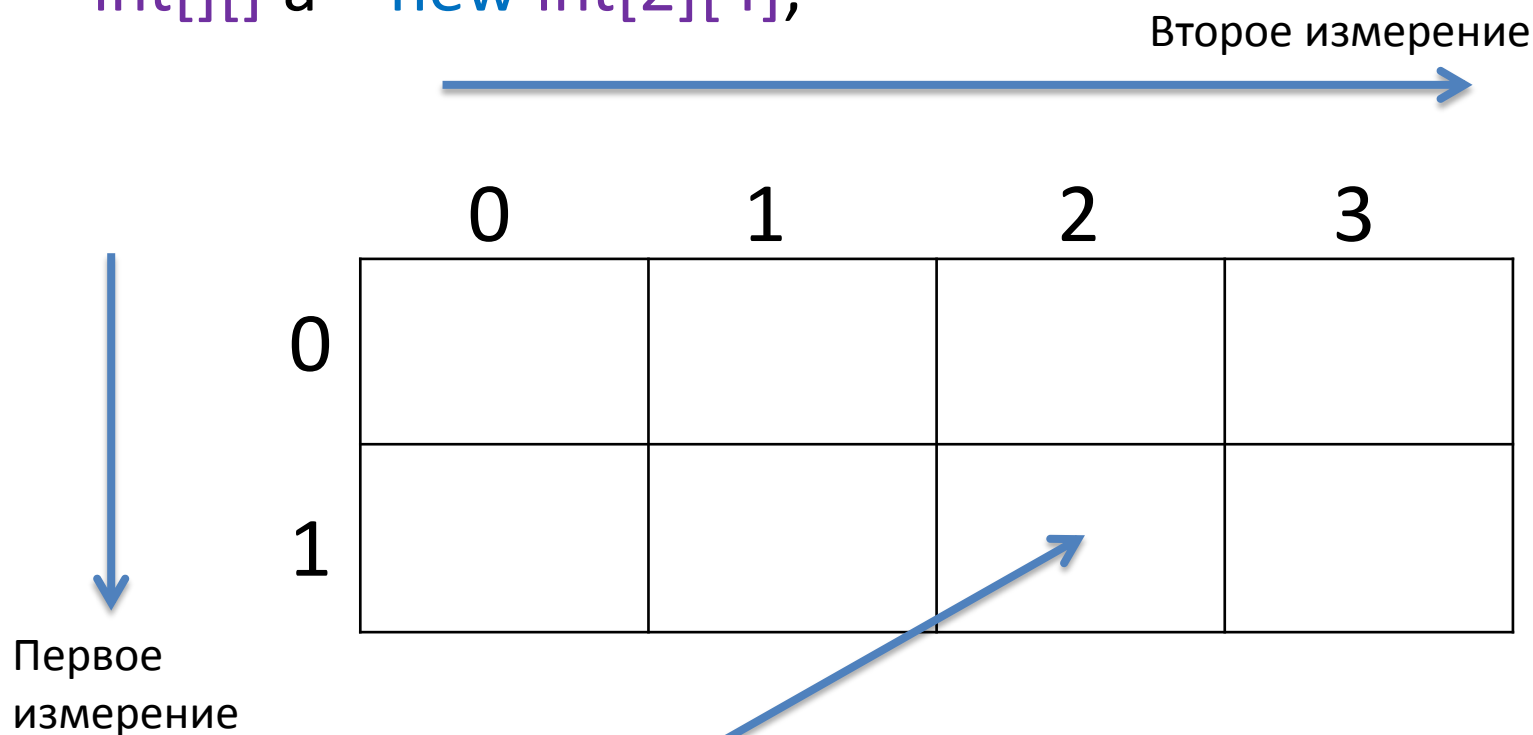


- `int x = a[1][2];`

Многомерные массивы

- Как многомерная таблица

- `int[][] a = new int[2][4];`



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

Многомерные массивы

- Для многомерных массивов также есть сокращенный синтаксис инициализации
- `int[][] a = { {1, 2, 3}, {2, 3, 4} };`

Стандартные функции

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

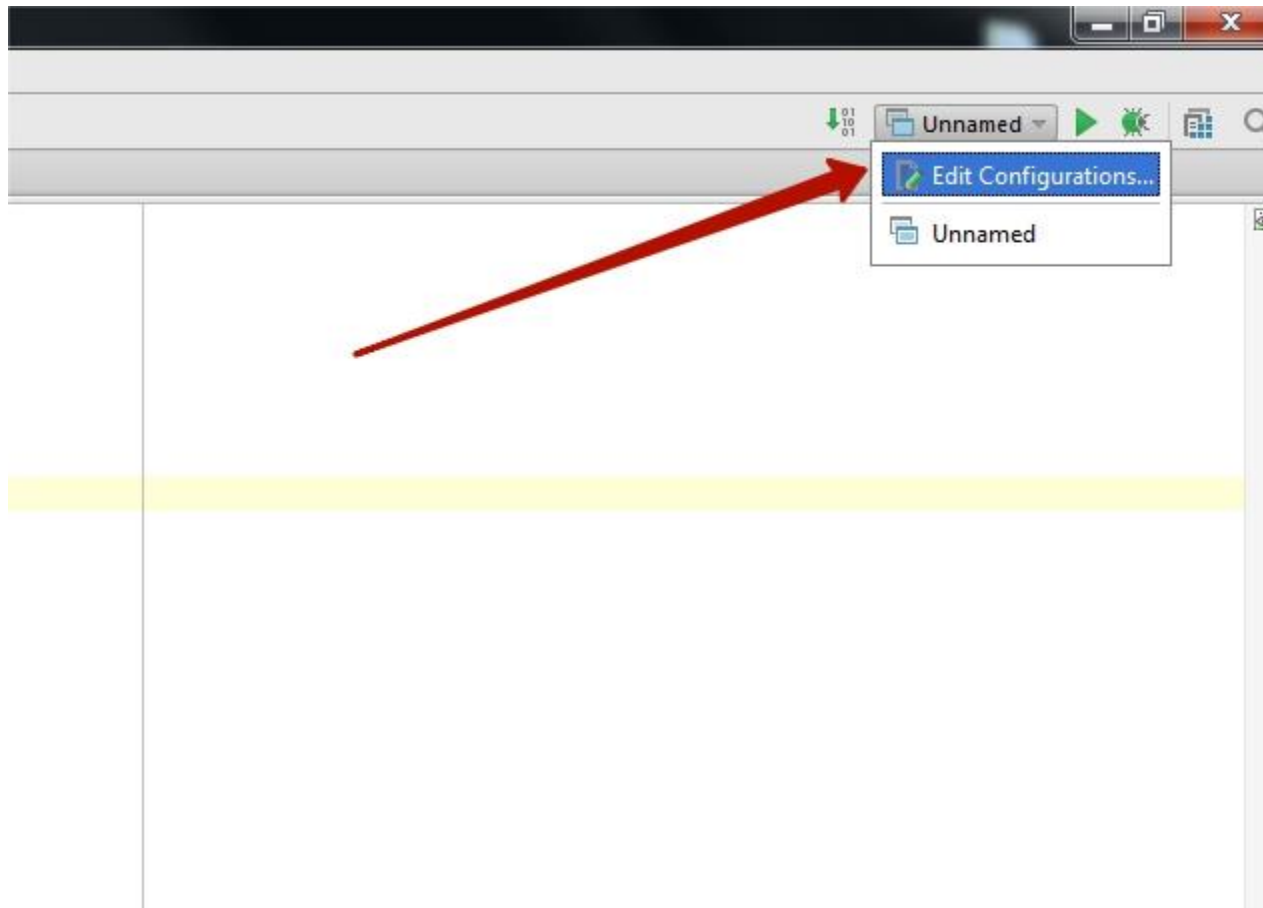
Печать массива

- Если распечатать массив, то не распечатается ничего осмысленного
- `int[] a = { 3, 5, 2 };`
`System.out.println(a);`
`// [I@1b6d3586`
- Поэтому если хочется распечатать массив, то придется использовать цикл по элементам
- Но есть удобная функция, которая преобразует массив в строку, использовать ее можно так:
- `System.out.println(Arrays.toString(a));`
`// [3, 5, 2]`

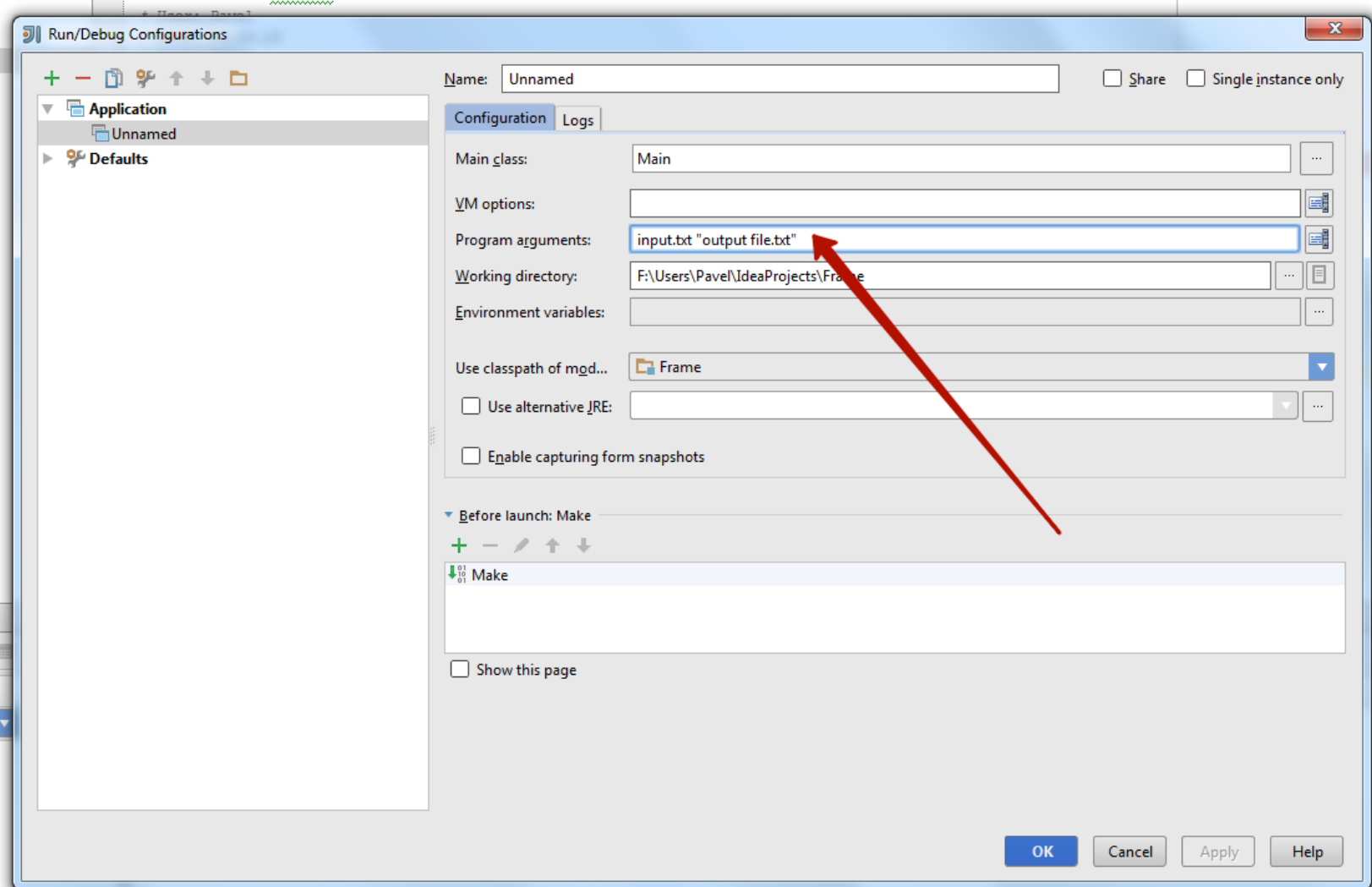
Аргументы программы

- 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

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

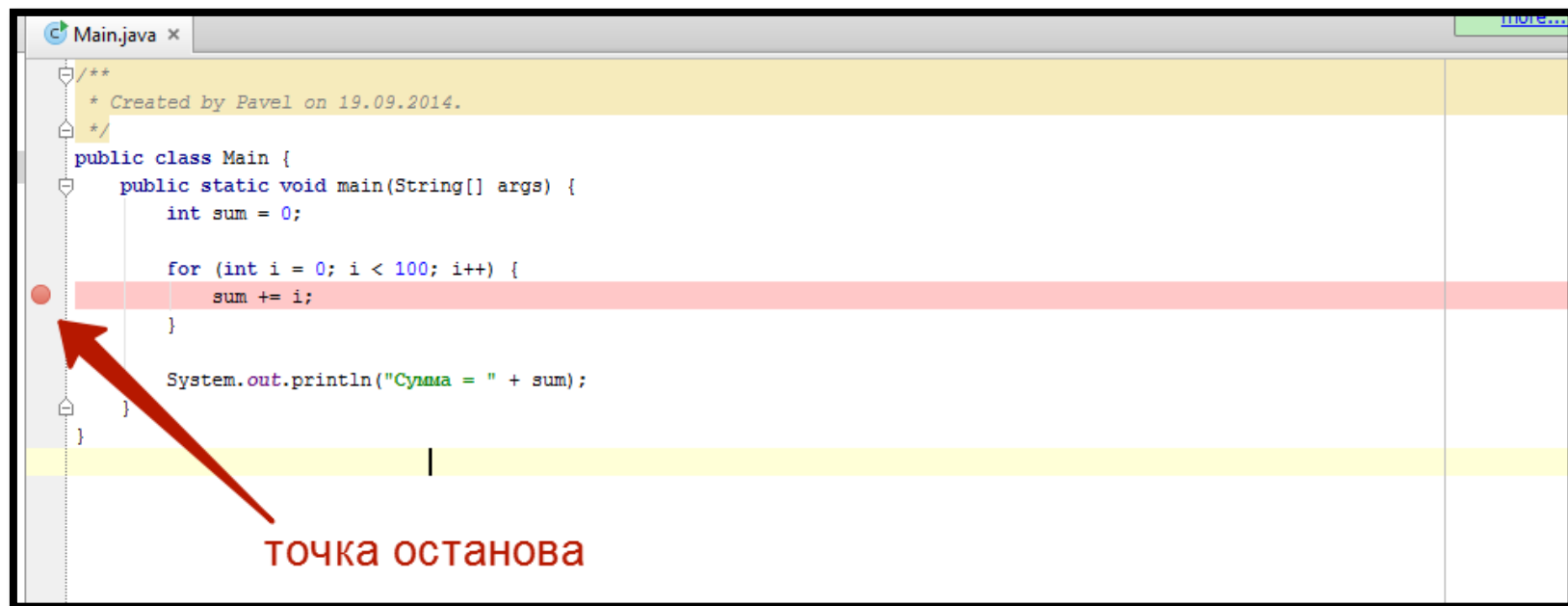
- Передать программе параметры
- Вывести в консоль количество параметров
- Вывести в консоль значения параметров при помощи цикла `foreach`

Отладка программ

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

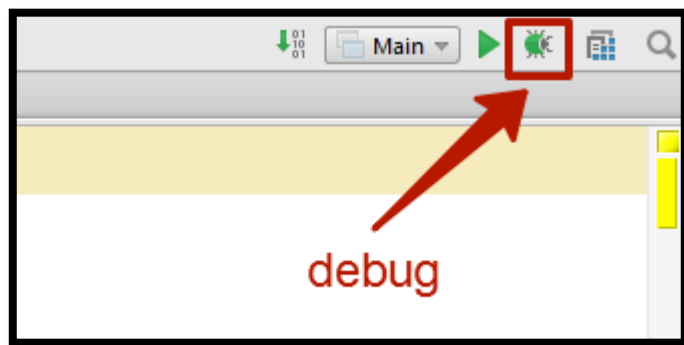
Точки останова

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



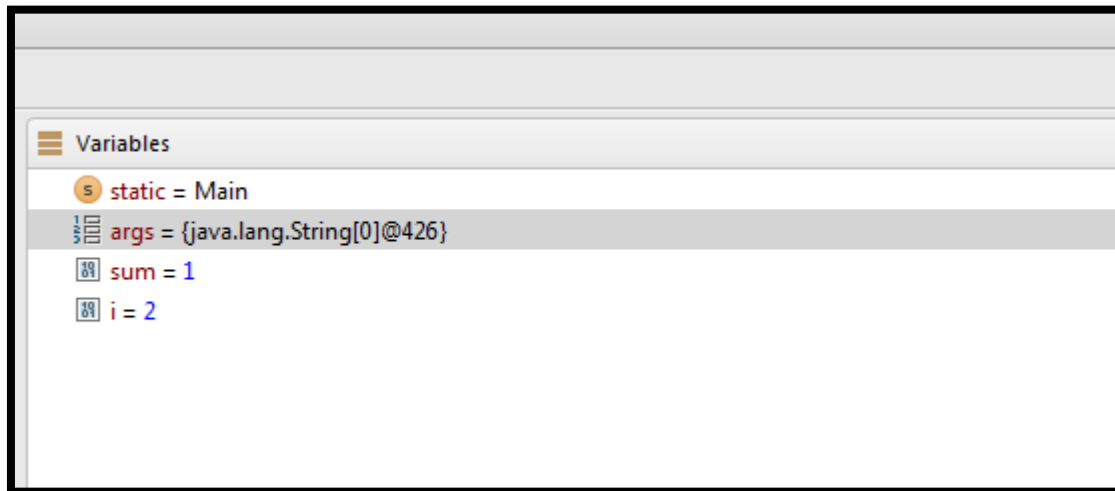
Запуск отладки

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



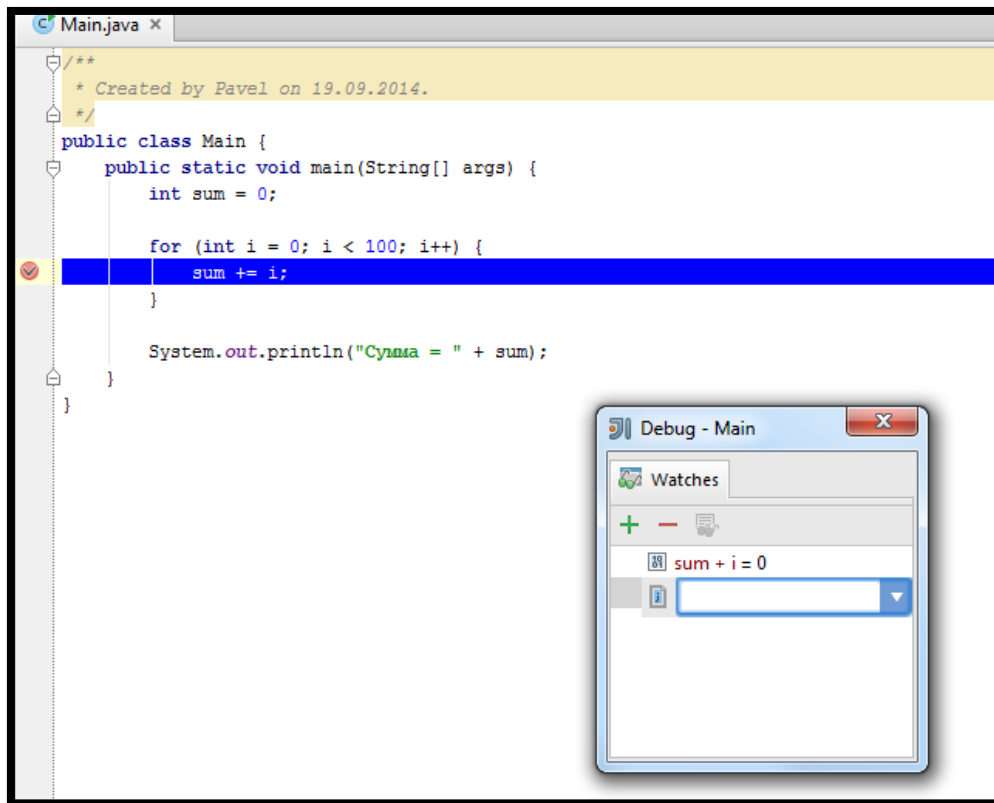
Просмотр значений переменных

- Когда программа остановлена, можно посмотреть текущие значения переменных



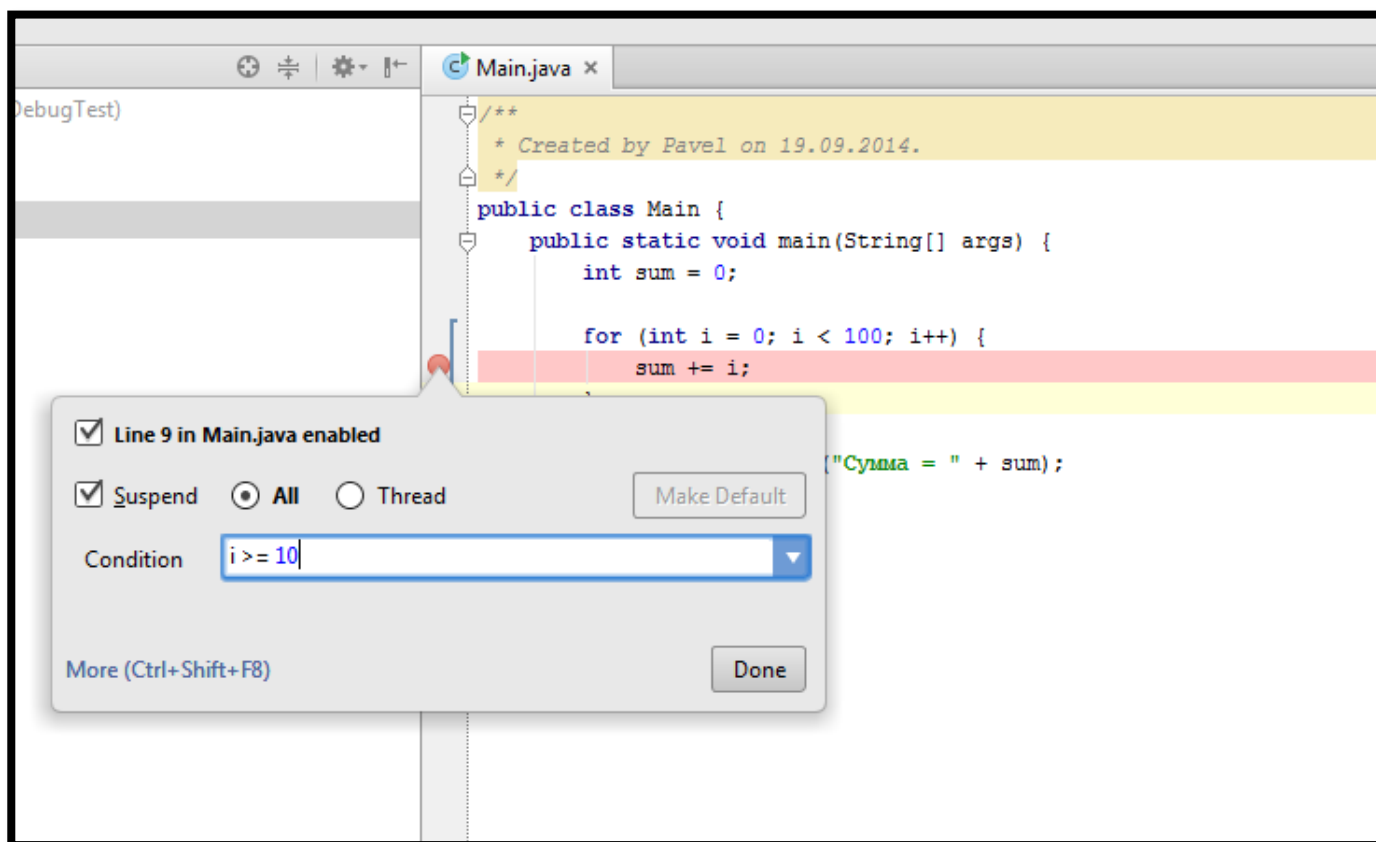
Просмотр результатов выражений

- 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

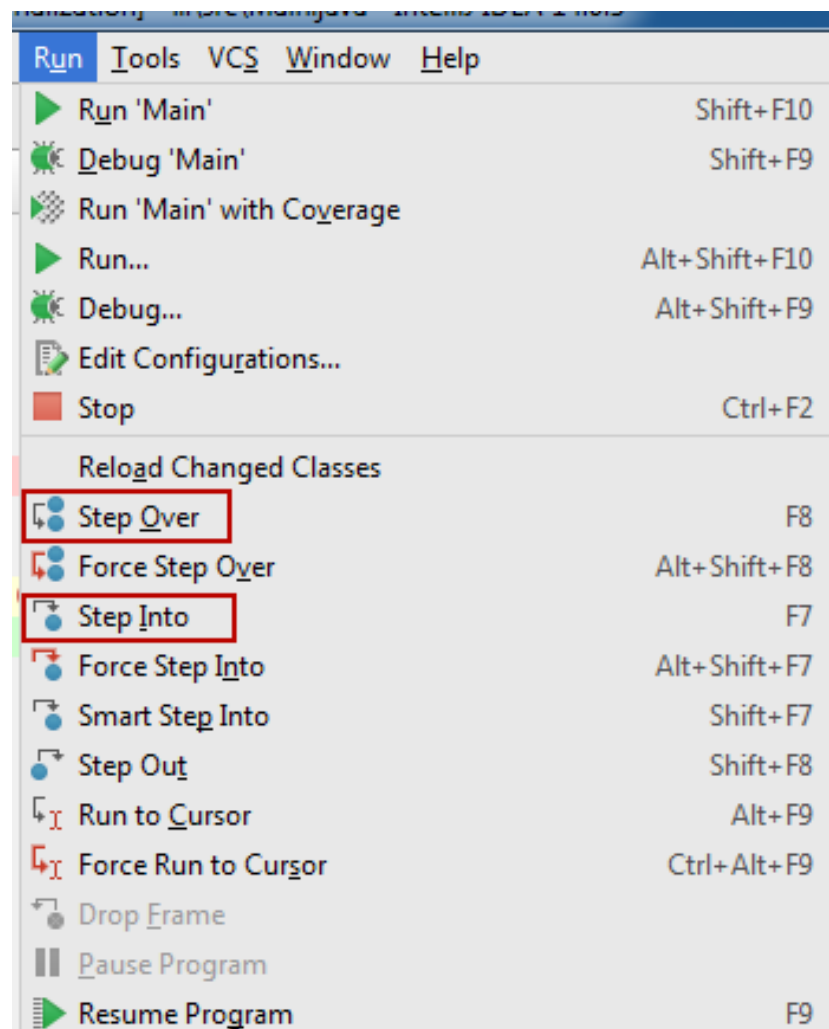


Пошаговая отладка

- 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

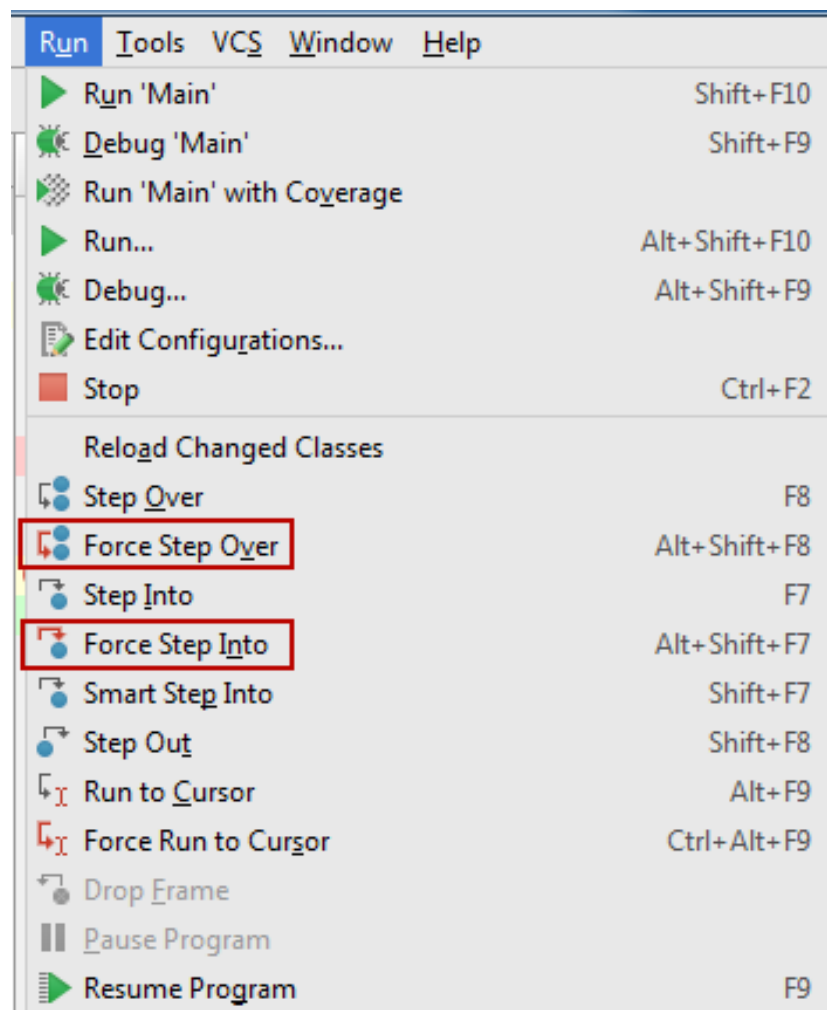
Пошаговая отладка

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



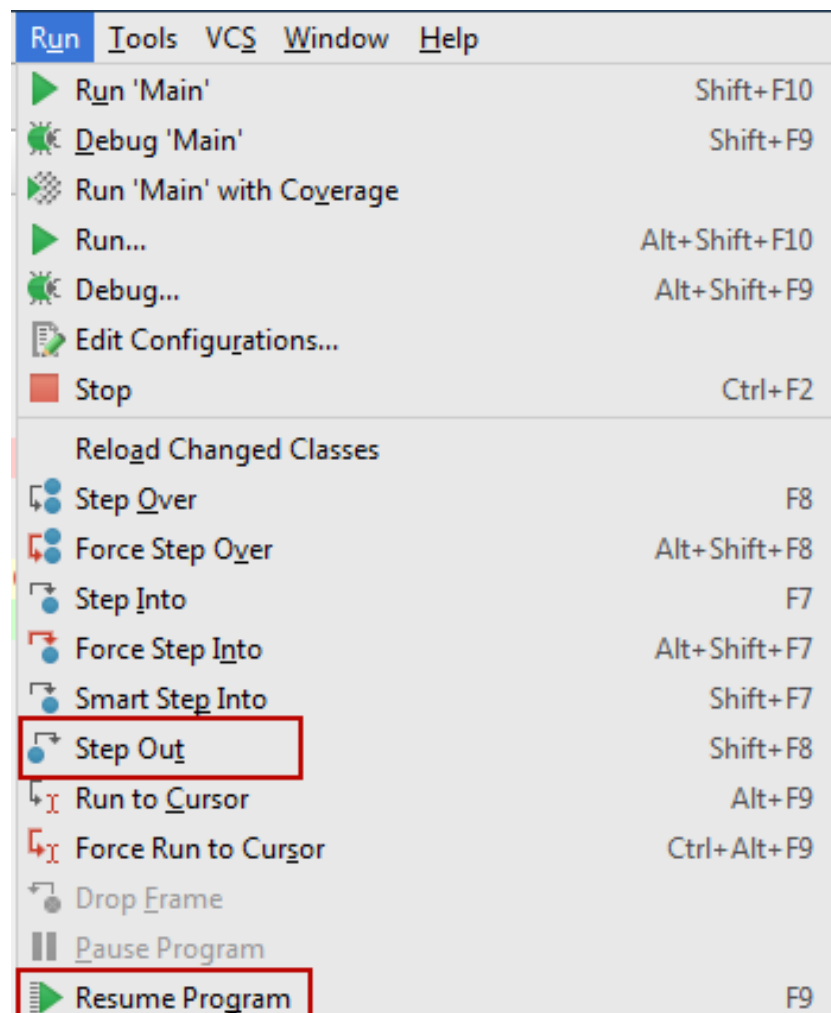
Пошаговая отладка

- 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



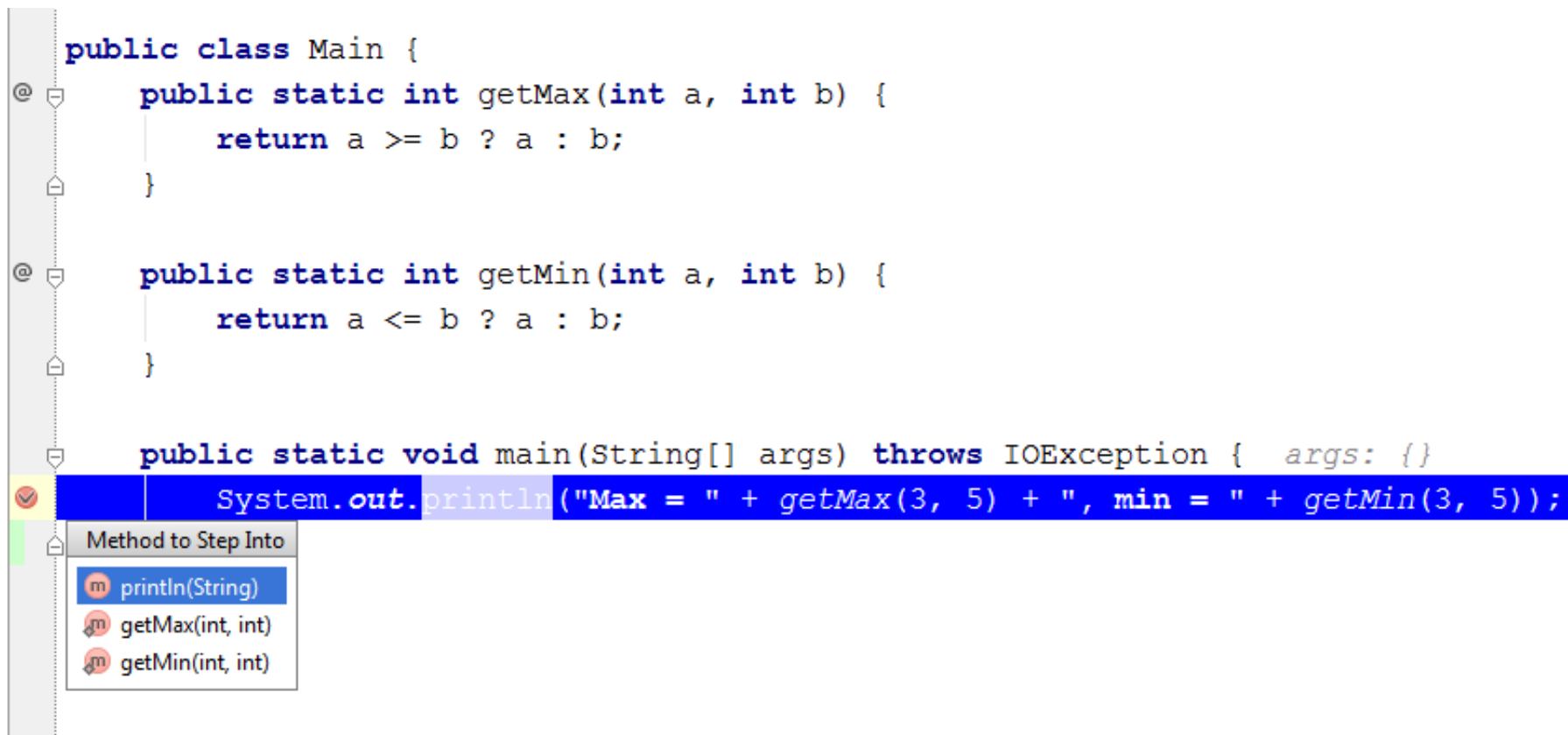
Пошаговая отладка

- 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));  
    }  
}
```

Method to Step Into

- println(String)
- getMax(int, int)
- getMin(int, int)

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 «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` `toUpperCase()`
- Пример:
`String s = "hello";`
`String b = s.toUpperCase(); // "HELLO"`

Задача на курс «Среднее арифм. массива»

- Найти среднее арифметическое элементов массива, которые являются четными числами

Задача на курс «Разворот массива»

- Переставить элементы массива в обратном порядке

Задача на курс «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`