Лекция 4.
Оператор switch.
Циклы while, do-while.
Отладка

Проверка на равенство вещ-х чисел

- Для вещественных чисел нельзя использовать проверку на равенство при помощи ==
- Это связано с ошибками округления – компьютер имеет ограниченную точность при работе с вещественными числами

- `System.out.println(2.0 - 1.1);`
  `// 0.8999999999999999`
- `System.out.println(2.0 - 1.1 == 0.9);`
  `// false`

Проверка на равенство вещ-х чисел

- Поэтому проверку на равенство нужно заменять на проверку, что число лежит в некотором небольшом диапазоне
- $a = b$
- $a - b = 0$

- Теперь позволяем разности отклоняться от 0 в диапазоне от $-\varepsilon$ до ε
- $-\varepsilon \leq a - b \leq \varepsilon$
- $|a - b| \leq \varepsilon$

Проверка на равенство вещ-х чисел

- $|a - b| \leq \varepsilon$

- В коде:
- double epsilon = 1.0e-10;

```
if (Math.abs(a - b) <= epsilon) {
    // a примерно равен b
}
```

- Команда Math.abs(x) выдает модуль вещественного числа
- В качестве epsilon можно брать любое маленькое положительное число

Другие проверки

- А как проверить, что два вещественных числа не равны?
- $|a - b| > \varepsilon$

- Как проверить, что одно число больше другого?
- $a - b > \varepsilon$

Задача

- Прочитать два вещественных числа с консоли
- Проверить, что они равны с учетом погрешности
- Выдать соответствующее сообщение

Константы

Константа

- **Константа** – это переменная, значение которой нельзя изменить

- В Java используется ключевое слово final:
- final int flatsOnFloorCount = 4;

- Константы нужны, чтобы нельзя было случайно (или специально) переприсвоить переменную, значение которой не должно меняться
- Пример: число Пи – Math.PI

Слово final

- Ключевое слово final означает, что данной переменной значение может быть присвоено только один раз, но не обязательно сразу же
- Пример:

```
public static void main(String[] args) {
    final int a = 5;
    a = 6;   // ошибка компиляции, присваиваем второй раз
    final int b;
    b = 3;   // OK
    b = 4;   // ошибка компиляции, присваиваем второй раз
}
```

Switch

Конструкция switch

```
switch (x) {
    case 0:
        System.out.println(0);
        break;
    case 1:
        System.out.println(1);
        break;
    default:
        System.out.println("иначе");
}
```

```
if (x == 0) {
    System.out.println(0);
} else if (x == 1) {
    System.out.println(1);
} else {
    System.out.println("иначе");
}
```

Case соответствует проверке в if
Default соответствует else

Конструкция switch

```
switch (x) {
    case 0:
        System.out.println(0);
        break;
    case 1:
        System.out.println(1);
        break;
    default:
        System.out.println("?");
}
```

Сначала вычисляется выражение в **switch()**, а затем результат последовательно сравнивается со значениями в **case**'ах

Если результат совпал, то выполняются команды, идущие после :

Каждую ветку **case** нужно завершать ключевым словом **break**. Оно означает, что выполнение блока **switch** закончится, и дальше будет исполняться код, идущий ниже **switch**'а

Конструкция switch

```
switch (x) {
    case 0:
        System.out.println(0);
        break;
    case 1:
        System.out.println(1);
        break;
    default:
        System.out.println("?");
}
```

switch может содержать необязательную ветку **default**

Она означает **else**, и выполнится если результат выражения не совпал ни с одним значением

Если в **switch** есть ветка **default**, то она обязана быть последней

Конструкция switch

- В каждой ветке case и default может быть много команд
- В case можно указывать только литералы и константы
- switch можно применять только для определенных типов:
  - Целые числа: byte, short, int (long нельзя)
  - Строки: String
  - Символы: char – пройдем позже в курсе
  - Енумы (Enums) – их будем проходить на курсе ООП
  - Классы-обертки: Character, Byte, Short, Integer – их будем проходить на курсе ООП
- Нельзя использовать вещественные числа

Конструкция switch

- Ветки в case не обязательно должны заканчиваться на break, и даже могут ничего не содержать
- Это позволяет выполнить одни и те же команды для нескольких разных значений
-

```
switch (number) {
    case 0:
    case 1:
        System.out.println("Это маленькие числа");
        break;
    case 2:
    case 3:
        System.out.println("А это большие");
}
```

Этот код выполнится если number равен 0 или 1

Этот код выполнится если number равен 2 или 3

Конструкция switch

- Но эту возможность следует использовать редко и очень осторожно
-

```
switch (number) {
    case 0:
    case 1:
        System.out.println("Это маленькие числа");
        break;
    case 2:
    case 3:
        System.out.println("А это большие");
}
```

Пусть произошло совпадение со значением, и выполнился блок кода, в котором не было break, как в ветке для case 1

В этом случае исполнение switch не прекратится, и будет выполняться код в последующих case, даже если результат не равен значениям в них, пока не встретится break, либо кончится весь switch

Задача

- Прочитать с консоли строку с названием команды
- Если ввели слово print, то прочитать с консоли еще одну строку, и напечатать ее
- Если ввели слово sum, то прочитать с консоли два вещественных числа, и вывести их сумму
- Если ввели что-то другое, то напечатать, что это неизвестная команда
- Использовать классический switch (без ->), warning игнорируем

Задача на дом «Switch»

- Прочитать с консоли три числа – два операнда и код команды
- Код команды должен быть от 1 до 4
- Если он равен 1, то выполнить сложение первых двух чисел. Если 2, то вычитание, если 3, то умножение, если 4, то деление.
- Если ввели число не от 1 до 4, то вывести, что неизвестная операция

- Использовать классический switch (без ->), warning игнорируем

Новый вариант switch

Новый вариант switch в Java

- Начиная с Java 12, появился новый вариант switch
- ```
  switch (x) {
      case 0 -> System.out.println(0);
      case 1 -> System.out.println(1);
      default -> System.out.println(“иначе”);
  }
  ```
- Этот вариант более краткий, здесь можно не писать break
- Вместо двоеточия используется стрелка ->
  ```

Несколько команд в ветке

- В ветках можно писать либо 1 команду без фигурных скобок, либо блок кода в фигурных скобках:
-
```
switch (x) {
    case 0 -> System.out.println(0);
    case 1 -> {
        System.out.println(1);
        System.out.println(2);
    }
    default -> System.out.println(“иначе”);
}
```

Несколько значений в case

- В case можно указывать несколько значений через запятую:
- ```
  switch (x) {
      case 0, 1, 2 -> System.out.println(“Малые числа”);
      case 3, 4 -> System.out.println(“Большие числа”);
  }
  ```
  ```

Новый вариант switch в Java

- Теперь switch является выражением, если все ветки содержат выражение:
-
```
int result = switch (x) {
    case 0 -> 1;
    case 1 -> 2 + 5;
    default -> 3;
};
```
- При этом ветки этого switch должны покрывать все возможные случаи
- Поэтому здесь нужен default

Новый вариант switch в Java

- Если в ветке несколько команд, то для выдачи результата из ветки нужно использовать ключевое слово yield:
- ```
  int result = switch (x) {
      case 0 -> 1;
      case 1 -> {
          int a = 5;
          yield a * 2;
      }
      default -> 3;
  };
  ```
  ```

Краткие операторы. Инкремент и декремент

Операции с присваиванием

- Можно писать

x += 1; вместо x = x + 1;

x -= 4; вместо x = x – 4;

- Аналогично существуют *=, /=, %=

Инкремент и декремент

- **Инкремент** – увеличение значения переменной на единицу
- **Декремент** – уменьшение значения переменной на единицу

- Эквивалентны:

x = x + 1;	x = x – 1;
x += 1;	x -= 1;
++x;	**--x;**
x++;	**x--;**

Постфиксный и префиксный варианты

- Вариант x++; и x--; называется **постфиксным**
- Вариант ++x; и --x; называется **префиксным**

- Разница между ними:
  - Префиксный вариант выполняет инкремент/декремент и выдает новое значение
  - Постфиксный вариант запоминает значение до инкремента/декремента, затем выполняет инкремент/декремент, а затем выдает запомненное старое значение

Постфиксный и префиксный варианты

```
int x = 4;

System.out.println(++x);
// выведет 5, новое значение

System.out.println(x++);
// 5, т.к. постфиксный оператор выдает старое
// значение

System.out.println(x);
// выведет 6
```

Вопрос

- Что выведет следующий код?
- ```
  int x = 30;
  int y = 5;
  System.out.println(x++ + y--);
  System.out.println(++x - ++y);
  System.out.println(x);
  System.out.println(y);
  ```
  ```

Циклы

Печать чисел от 1 до 100

- ```
  System.out.println(1);
  System.out.println(2);
  …
  System.out.println(99);
  System.out.println(100);
  ```

- Как напечатать числа от 1 до n, если n мы читаем с консоли?
  ```

Циклы

- **Цикл** – это конструкция языка, которая позволяет выполнять один и тот же блок кода много раз, пока выполняется некоторое условие

- В Java существует 4 вида циклов:
  - while
  - do-while
  - 2 разновидности цикла for

Цикл while

- while (логическое выражение)
  инструкция

- Как работает:
  - Шаг 1. Вычисляется значение логического выражения **(условие цикла)**
  - Шаг 2. Если оно ложно, то цикл завершается
  - Шаг 3. Если оно истинно, то выполняется **тело цикла (инструкция)**. Затем переход на шаг 1

Сумма чисел от 0 до 9

- Часто цикл используют, чтобы пройтись по диапазону чисел

- ```
  int i = 0;                    // счетчик цикла
  int sum = 0;

  while (i <= 9) {
      sum += i;
      ++i;
  }

  System.out.println("Сумма = " + sum);
  ```
  ```

Названия счетчиков

- Для названий переменных-счетчиков цикла часто используют короткие имена, состоящие из одной буквы
- Общепринято называть переменную-счетчик буквой i
- Если имя i уже занято, то использовать j,k,m,n и так далее
- Букву l пропускают, т.к. она похожа на 1

Термины циклов

- while (логическое выражение)
  инструкция

- Логическое выражение называется **условием цикла** (или **условием продолжения цикла**)
- Инструкция называется **телом цикла** – код, который выполняется внутри цикла
- Одно выполнение тела цикла называется **итерацией**

Задача

- Найти сумму чисел от 0 до 9
- Переделать программу, чтобы найти сумму от 3 до 21 включительно
- Переделать программу, чтобы найти сумму только четных чисел от 3 до 21 включительно
- Дополнительно найти количество четных чисел от 3 до 21 включительно

Цикл while

- Тело цикла может не выполниться **ни разу**, если условие сразу было ложным
- Если условие всегда истинно, то цикл выполняется бесконечно. Это называется **зацикливанием** и обычно является ошибкой
- В примере зацикливание может произойти если забыть сделать ++i;

Задача на дом «Среднее арифметическое»

- Написать программу, вычисляющую среднее арифметическое чисел из некоторого диапазона чисел (например, от 3 до 17)
- Концы диапазона задать переменными, начальное число берите > 1, чтобы было посложнее
- **Среднее арифметическое чисел –** нужно сумму всех чисел поделить на количество этих чисел

- В этом же классе - найти среднее арифметическое только четных чисел из этого диапазона чисел

Задача на курс «Числа Фибоначчи»

- Написать программу, которая принимает с консоли целое число n и возвращает число Фибоначчи с номером n.
- Числа Фибоначчи задаются следующим образом:
- $f_o = 0, f_1 = 1, f_n = f_{n-1} + f_{n-2}$

Цикл do-while

- do
  инструкция
  while (логическое выражение);
- Как всегда, инструкция – 1 команда или блок кода в фигурных скобках
- Как работает:
  - Шаг 1. Выполняется тело цикла
  - Шаг 2. Проверятся условие. Если истинно, то возвращаемся на шаг 1. Если ложно, то конец цикла

Цикл do-while: сумма чисел от 0 до 100

```
int i = 0;
int sum = 0;

do {
    sum += i;
    ++i;
} while (i <= 100);

System.out.println("Сумма = " + sum);
```

Цикл do-while

- Отличие от while: тело цикла do-while всегда выполняется хотя бы 1 раз, т.к. первая проверка условия происходит после 1 итерации цикла

Задача на дом «Do-while»

- Сделать задачу про среднее арифметическое с циклом do-while

- Эту задачу нужно присылать в одну цепочку писем с версией с while

Задача на дом «Сумма ряда»

- Написать программу, которая находит результат такого выражения:
  - 1 – 4 + 9 – 16 + 25 – 36 ...
- Количество чисел в этом выражении должно быть параметром программы

Задача на дом «10 чисел в строке»

- Распечатать числа от 1 до 100 при помощи цикла while, но выводить по 10 чисел в строке, дальше делать перевод строки
- 1 2 3 4 5 6 7 8 9 10
  11 12 13 14 15 16 17 18 19 20
  …

- Сложная версия задачи:
  - Выводить числа ровно, чтобы они были друг под другом. Использовать System.out.printf
  - Возможность задать начальное и конечное число, и по сколько чисел в строке выводить

Задача на дом «Цифры числа»

- Прочитать с консоли целое число
- Найдите сумму его цифр
- Найдите сумму только тех цифр числа, которые являются нечетными числами
- Найдите максимальную цифру числа

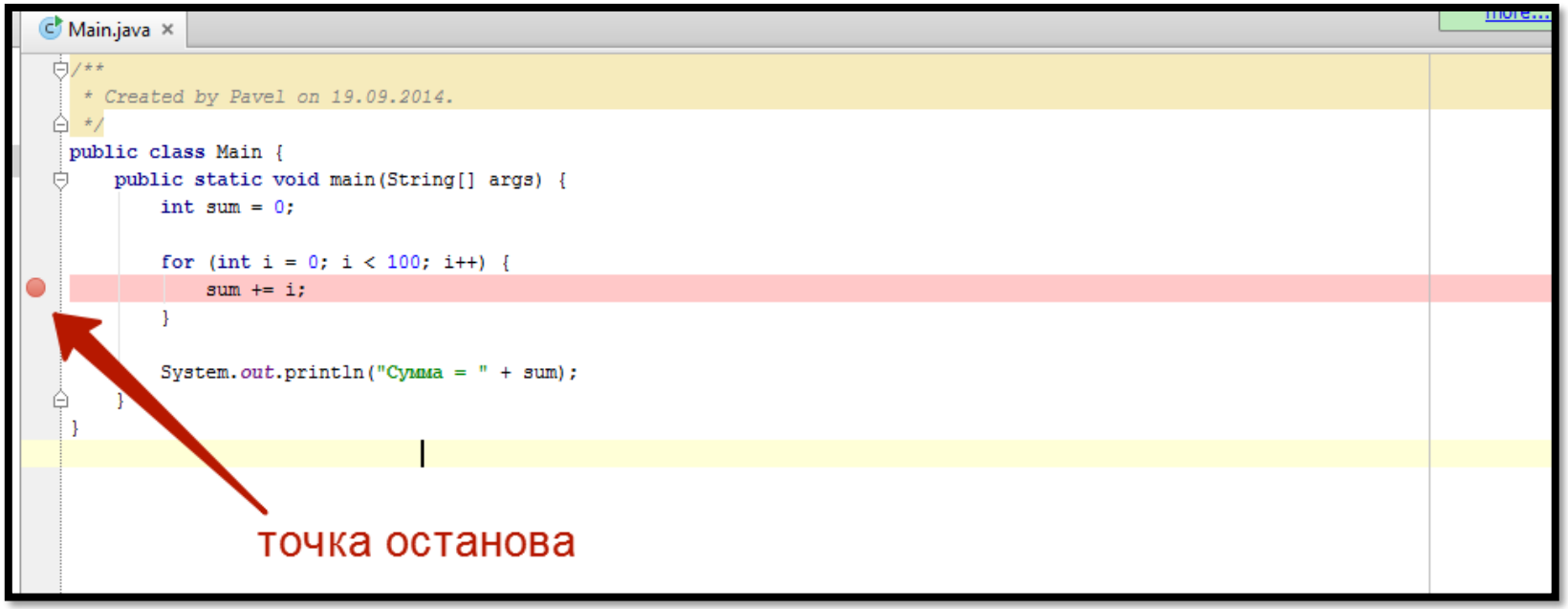
Отладка программ

Отладка программ

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

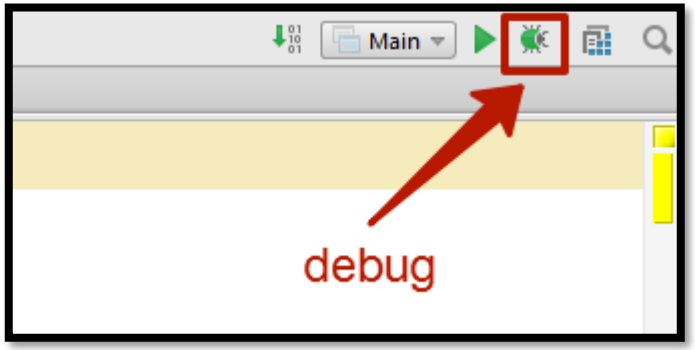
Точки останова

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



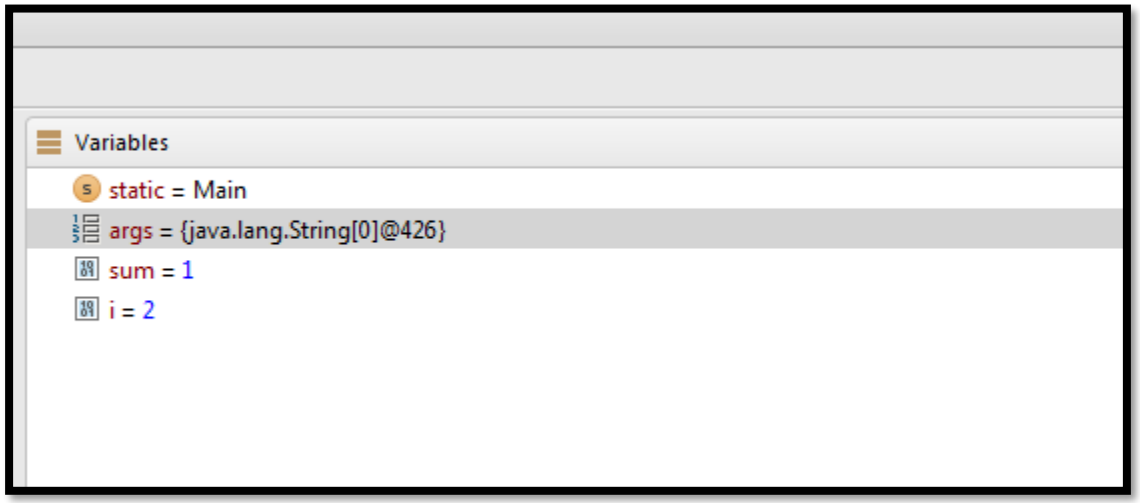
Запуск отладки

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



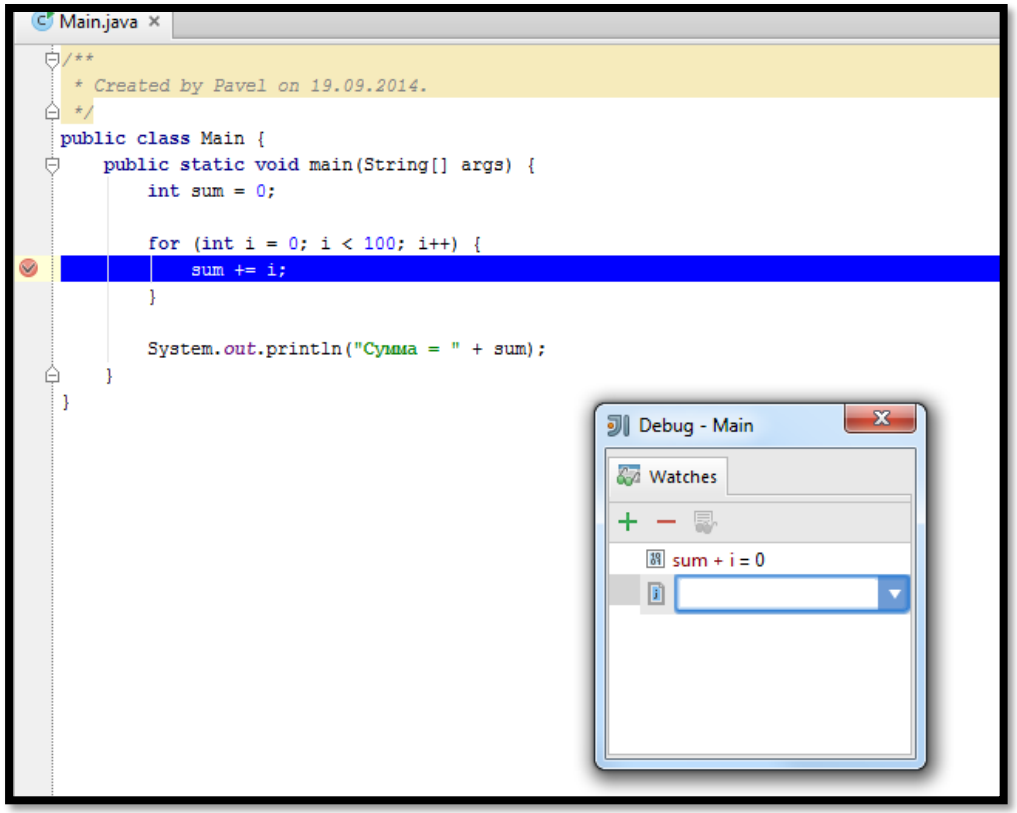
Просмотр значений переменных

- Когда программа остановлена, можно смотреть текущие значения переменных



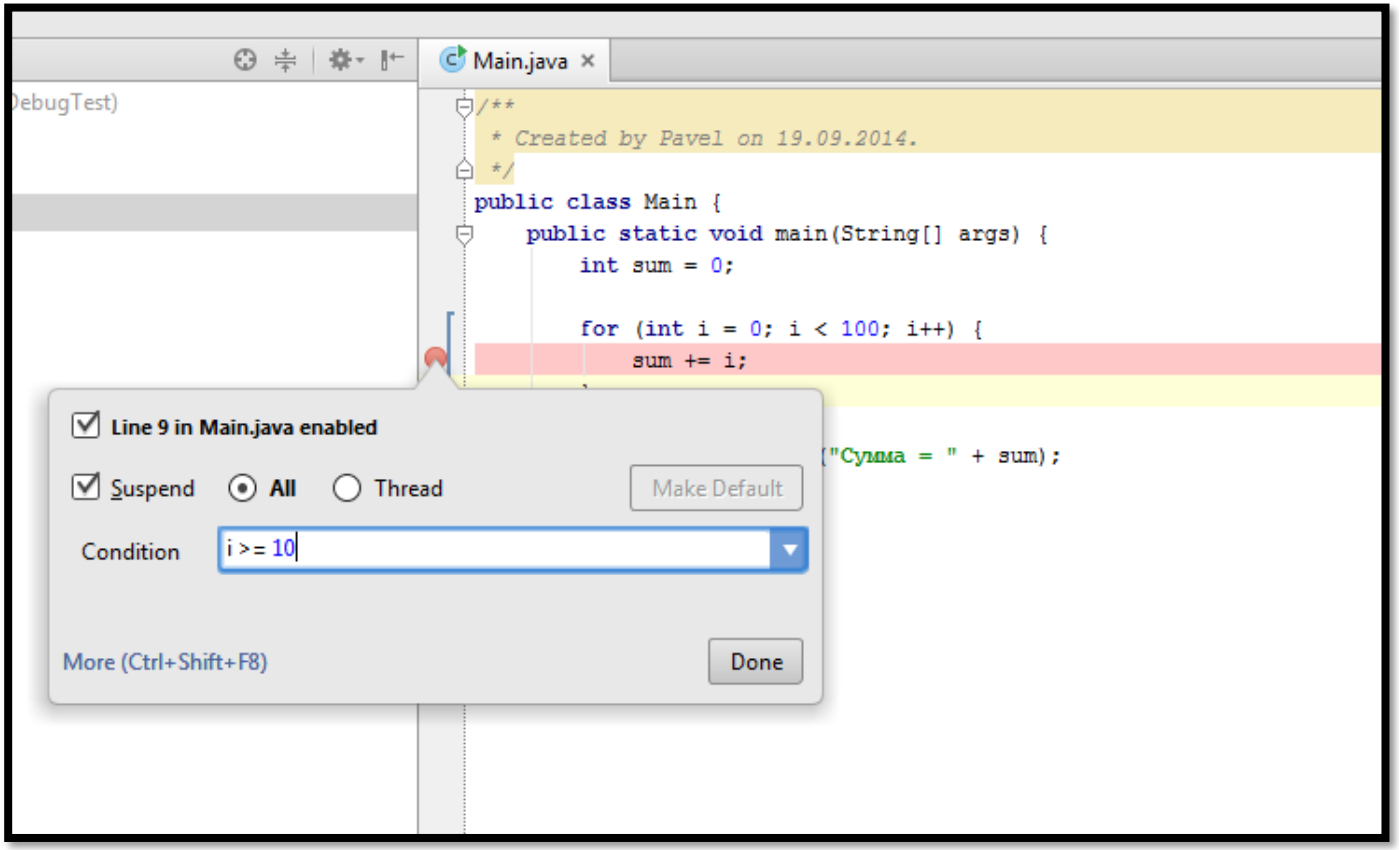
Просмотр результатов выражений

- Когда программа остановлена, можно смотреть 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