

# Циклы

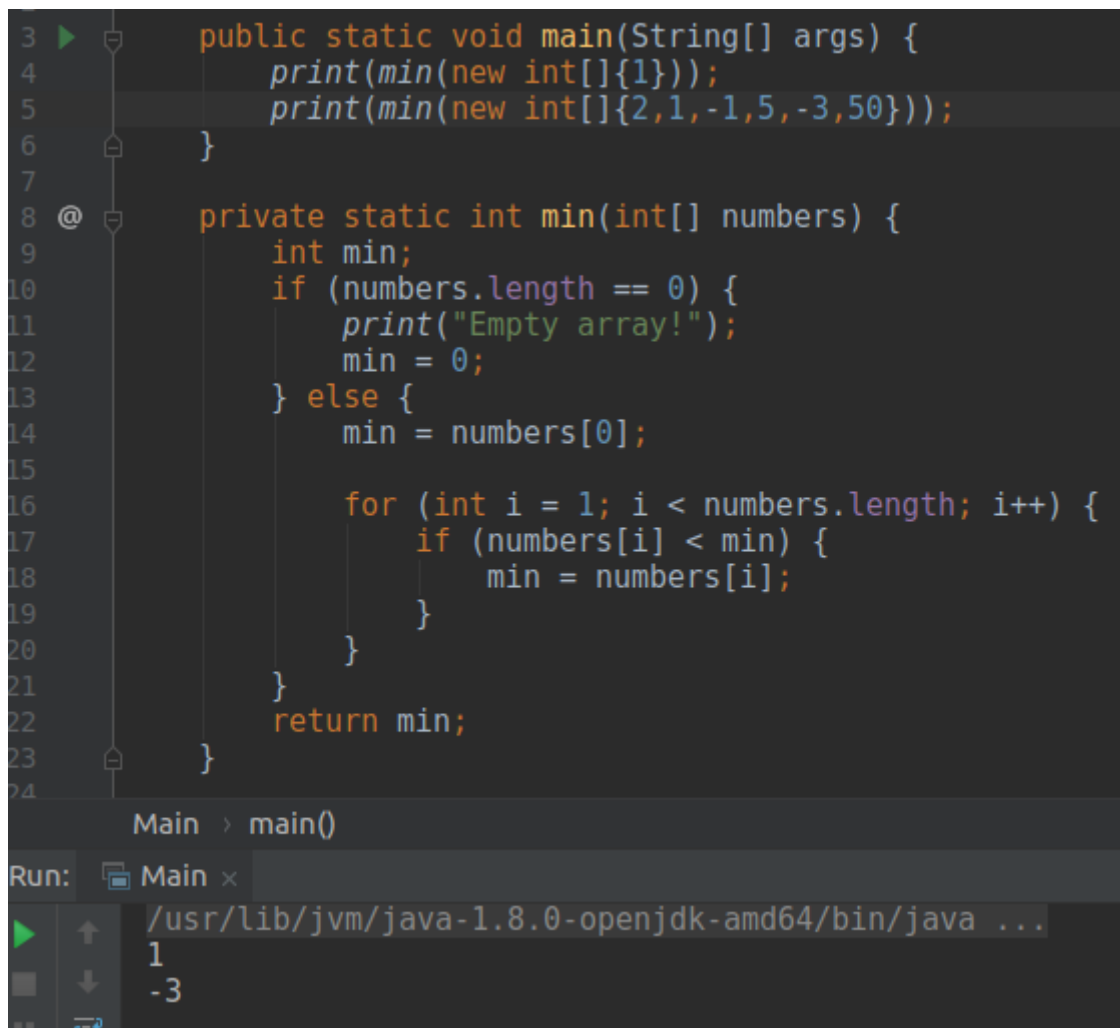
## Повторение и прерывание

### Содержание

1. Прерывание цикла. foreach
2. Разные конфигурации цикла for

### 1. Прерывание цикла

Давайте для начала выполним пару задач из предыдущей лекции (номер 13).



```
3 public static void main(String[] args) {
4     print(min(new int[]{1}));
5     print(min(new int[]{2,1,-1,5,-3,50}));
6 }
7
8 private static int min(int[] numbers) {
9     int min;
10    if (numbers.length == 0) {
11        print("Empty array!");
12        min = 0;
13    } else {
14        min = numbers[0];
15
16        for (int i = 1; i < numbers.length; i++) {
17            if (numbers[i] < min) {
18                min = numbers[i];
19            }
20        }
21    }
22    return min;
23 }
24
```

Run: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...

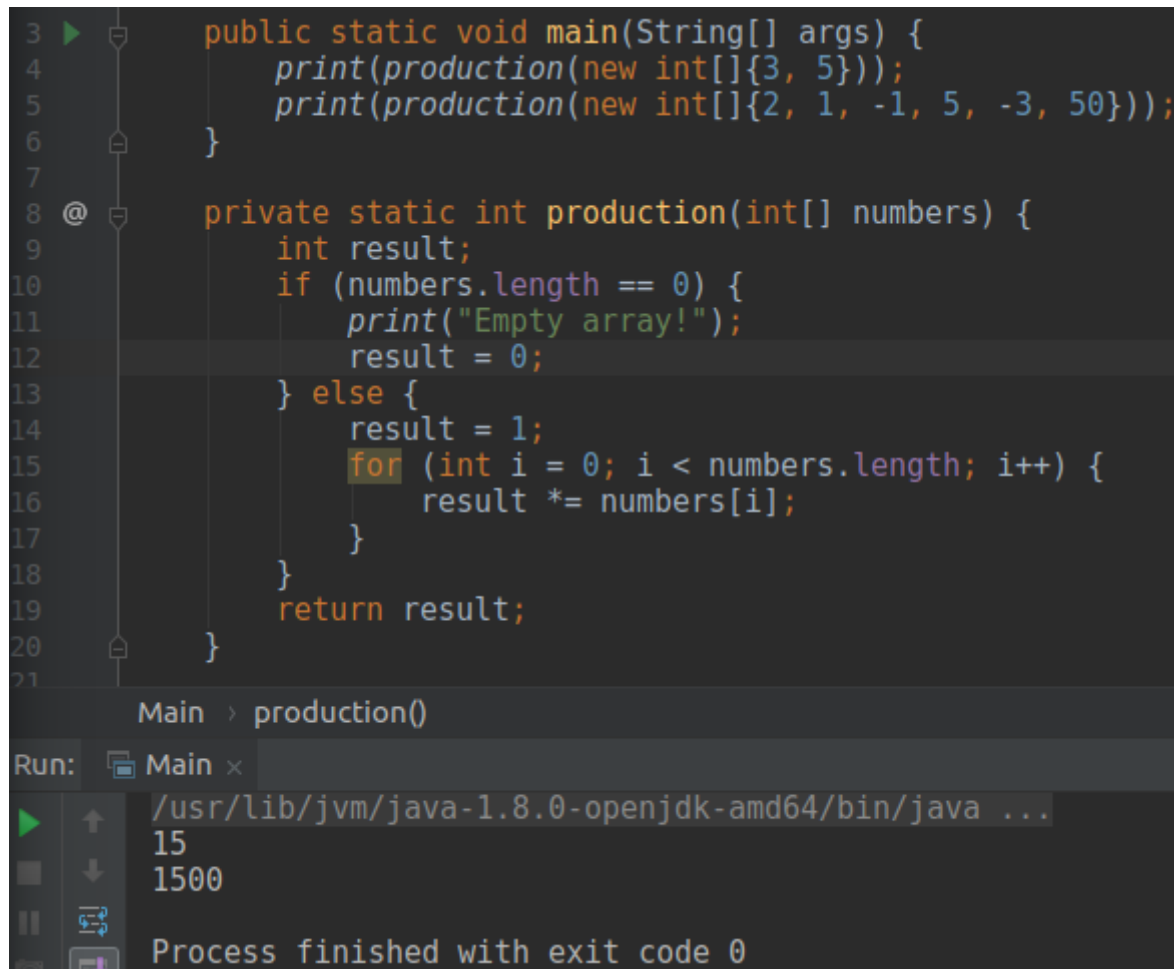
1

-3

Задача номер 1 из 13-ой лекции – найти минимум из любого количества чисел. Сначала проверим входной аргумент на пустоту и после уже найдем минимум. Мы предполагаем что первое число в массиве и есть минимум и потому уже сравниваем его со следующим и если следующее оказалось меньше первого, то перезаписываем наш буффер(переменная min). Мы понимаем, что нам не нужно писать цикл с нуля, потому что сравнивать первое число само с

собой не нужно. Именно поэтому у нас цикл `i=1`. Так что вам не всегда нужно будет писать `i=0; i<n; i++`. Ниже мы увидим, что значения могут быть какие угодно. Кстати, для пустого массива просто уберите единицу из линии 4.

Рассмотрим умножение (задача номер 2, лекция 13). Здесь мы возьмем переменную с единицей, чтобы умножать ее на все числа массива. Так удобнее и я покажу почему. Смотрим



```
3 public static void main(String[] args) {
4     print(production(new int[]{3, 5}));
5     print(production(new int[]{2, 1, -1, 5, -3, 50}));
6 }
7
8 @ private static int production(int[] numbers) {
9     int result;
10    if (numbers.length == 0) {
11        print("Empty array!");
12        result = 0;
13    } else {
14        result = 1;
15        for (int i = 0; i < numbers.length; i++) {
16            result *= numbers[i];
17        }
18    }
19    return result;
20 }
21
```

Main > production()

Run: Main x

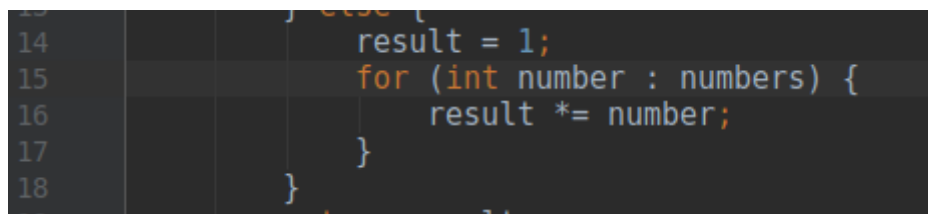
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...

15

1500

Process finished with exit code 0

Когда мы проходим циклом по всем элементам массива, то наша среда разработки выделяет слово `for`. Давайте посмотрим что она предлагает сделать. Ставим курсор на него и жмем `Alt+Enter`. Среда разработки предлагает заменить обычный `for` на `foreach`. Давайте посмотрим что будет.



```
14         result = 1;
15         for (int number : numbers) {
16             result *= number;
17         }
18     }
19     return result;
```

Вместо того, чтобы объявлять итератор мы объявили число `int number`, и каждый раз когда мы находимся на отрезке от нуля до конца массива в эту переменную автоматически будет присваиваться значение. т.е. на первой итерации будет `number = numbers[0]`, на второй: `number = numbers[1]`. Сути не меняет, но читается намного проще. Когда же нам использовать обычный `for`, а когда `foreach`? Если вам нужен индекс, вам важно знать на какой итерации вы

находитесь, то используйте `for (int i=`. Иначе же если вам все равно – то по возможности используйте `foreach`.

Ок, мы облегчили нам жизнь. Но этого недостаточно. Давайте вспомним, что если одно из чисел умножения равно нулю, то и произведение будет ноль. Мы бы могли сначала проверить, есть ли среди чисел ноль, и только потом производить вычисления. Но так будет больше работы. Представьте, вы проходите по всему массиву циклом и ищете ноль, а потом, если его нет, то еще раз проходите весь массив и перемножаете все числа. Было бы классно прервать выполнение цикла в нужный момент. Давайте посмотрим на такой код.

```
3 public static void main(String[] args) {
4     print(production(new int[]{3, 5}));
5     print(production(new int[]{2, 1, 0, -1, 5, -3, 50}));
6 }
7
8 private static int production(int[] numbers) {
9     int result;
10    if (numbers.length == 0) {
11        print("Empty array!");
12        result = 0;
13    } else {
14        result = 1;
15        for (int number : numbers) {
16            if (number == 0) {
17                result = 0;
18                print("Zero iz found");
19                break;
20            } else {
21                result *= number;
22            }
23        }
24    }
25    return result;
26 }
```

Main > production()

Run: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...

15

Zero iz found

0

Если число равно нулю, то присвоить результату ноль и прервать цикл (ключевое слово `break`). Чтобы быть уверенным выводим в консоль строку. Но что если мы все равно не верим в то, что цикл прервался? Как мы можем проверить? Мы можем выводить просто каждое число и увидеть. Или же подебажить. Поставить брейкпойнт на линии 16 и 17 и 21 и 25. И посмотреть куда идет компилятор.

Прерывание цикла очень полезная штука, ведь предположим у вас стоит задача – содержит ли массив число ноль? Вы просто пройдетесь по циклу и когда найдете его, то прервете выполнение и выйдете с ответом да. Ведь предположим у вас массив из 5 миллионов

элементов. Сколько нужно времени для прохождения по массиву? И что если первое же число ноль. Вообще-то мы можем дать ответ на этот вопрос!

```
3  public static void main(String[] args) {
4      int size = 5_000_000;
5      int[] array = new int[size];
6      for (int i = 0; i < size; i++) {
7          array[i] = i + 1;
8      }
9      array[1000] = 0;
10
11      print(containsZero(array));
12  }
13
14  @ private static boolean containsZero(int[] numbers) {
15      final long timeStart = System.currentTimeMillis();
16      int zeroCount = 0;
17      if (numbers.length == 0) {
18          print("Empty array!");
19      } else {
20          for (int number : numbers) {
21              if (number == 0) {
22                  zeroCount++;
23              }
24          }
25      }
26      long timeEnd = System.currentTimeMillis();
27      print("duration: " + (timeEnd - timeStart));
28      return zeroCount > 0;
29  }
```

Main > containsZero()

Run: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...  
duration: 7  
true

Process finished with exit code 0

Создаем массив на 5 миллионов чисел. Туда через цикл кладем значения. Тысячное число перезапишем на ноль. Класс System может дать нам время в миллисекундах. Вызываем его в начале метода и в конце и разницу выведем на экран. Заметьте, для начала мы не прерываем цикл. И мы видим, что прохождение и проверка каждого значения из 5 миллионов чисел заняло 7 миллисекунд (на вашем компьютере может быть меньше или больше в зависимости от мощностей). А теперь давайте добавим break после 22-ой линии.

Видим ноль миллисекунд. Т.е. почти мгновенно. Ведь мы прошли не по 5 миллионам чисел, а по тысяче.

Чтобы убедиться в правильности наших суждений, давайте нулевой элемент поставим на 4900000

```
20         for (int number : numbers) {
21             if (number == 0) {
22                 zeroCount++;
23                 break;
24             }
        }
    }

Main > containsZero()

Run: Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
duration: 0
true
```

```
7         array[i] = i + 1;
8     }
9     array[49000000] = 0;
10
11     print(containsZero(array));
12 }
13
14 @ private static boolean containsZero(int[] numbers) {
15     final long timeStart = System.currentTimeMillis();
16     int zeroCount = 0;
17     if (numbers.length == 0) {
18         print("Empty array!");
19     } else {
20         for (int number : numbers) {
21             if (number == 0) {
22                 zeroCount++;
23                 break;
24             }
        }
    }
}

Main > main()

Run: Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
duration: 6
true

Process finished with exit code 0
```

Видим закономерные 6 миллисекунд. Так что все по честному. И здесь кто-то скажет, что разница в 7 миллисекунд не такая и большая. Да, если вы ищете ноль среди чисел. Но реальность оказывается сложнее. Бывает вы сравниваете сложные объекты (об этом чуть позже, сразу как закончим с синтаксисом) и даже если их сотня, то это уже сложнее. Так что не забывайте про break;

## 2. Разные конфигурации цикла for

И напоследок решим задачу 6, она сложнее других.

Арифметическая прогрессия это когда разница между каждым следующим членом прогрессии постоянна. Т.е. разница между третьим членом и вторым равна разнице между

вторым и первым. Для этого мы возьмем и посчитаем разницу между вторым и первым членами. После этого нам нужно проверить, что эта разница одна и та же. Обратите внимание на цикл. Мы начинаем с индекса 1, т.е. со второго элемента, но не идем до конца, а идем до предпоследнего индекса. Ведь мы сравниваем элемент с индексом и следующим. Вы можете написать иначе – например `for (int i=2, i<numbers.length;i++) { int diff = numbers[i] – numbers[i-1].` Суть одна и та же.

```
3 public static void main(String[] args) {
4     print(isProgressive(new int[]{5, 8, 11, 14, 17}));
5     print(isProgressive(new int[]{4, 8, 10, 12}));
6 }
7
8 private static boolean isProgressive(int[] numbers) {
9     boolean result = true;
10    if (numbers.length == 0) {
11        print("Пустой массив!");
12        result = false;
13    } else if (numbers.length > 2) {
14        int difference = numbers[1] - numbers[0];
15        for (int i = 1; i < numbers.length - 1; i++) {
16            int diff = numbers[i + 1] - numbers[i];
17            if (diff != difference) {
18                result = false;
19            }
20        }
21    } else {
22        print("слишком мало данных");
23        result = false;
24    }
25    return result;
26 }
27
```

Main > main()

Run: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...

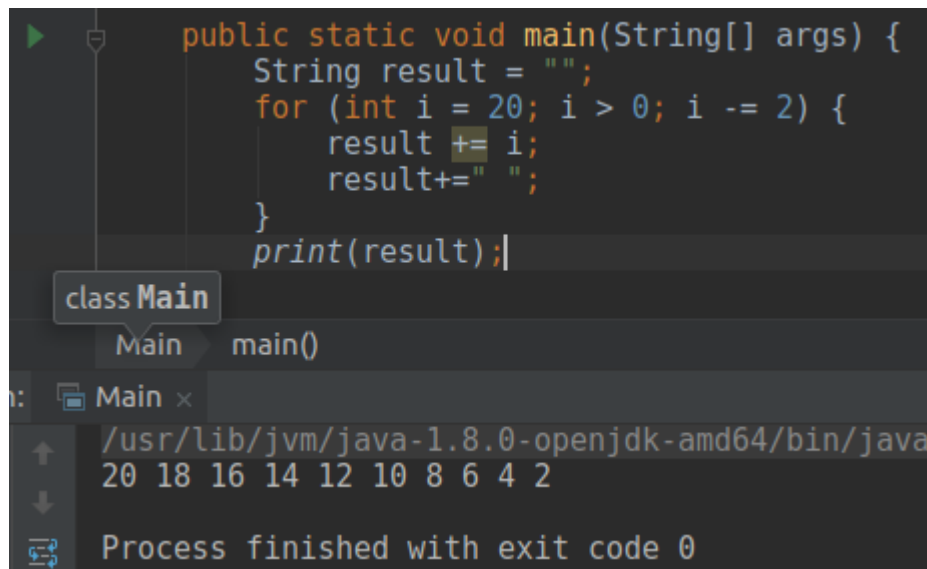
true

false

Process finished with exit code 0

И рассмотрим последний вопрос – а что если нам нужно например не инкрементировать итератор каждый раз, а например уменьшать по 2 и начинать не с нуля, а с 20 например? Это тоже можно сделать легко и просто. Мы стартуем с 20 и при каждой итерации уменьшаем на 2 и доходим до нуля.

У вас может быть более сложное условие нежели `i>0` или `i<0`. Это неважно. Главное сохранять структуру `for` (итератор, условие, изменение)



The screenshot shows an IDE with a Java class named `Main`. The `main` method contains a loop that calculates the sum of even numbers from 20 down to 2, with spaces between each number. The output window shows the result: `20 18 16 14 12 10 8 6 4 2`. The process finished with exit code 0.

```
public static void main(String[] args) {
    String result = "";
    for (int i = 20; i > 0; i -= 2) {
        result += i;
        result += " ";
    }
    print(result);
}
```

class Main

Main main()

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java

20 18 16 14 12 10 8 6 4 2

Process finished with exit code 0

Напоследок несколько задач для закрепления темы

1. Метод выводит все числа кратные 3 от нуля до какого-то числа (входной параметр)
2. Метод выводит все числа кратные числу, которое передаем в аргументе
3. Вывести первые  $n$  членов последовательности Фибоначчи (сами погуглите что это)
4. Найти минимум и максимум в массиве и вывести разницу
5. Найти в массиве повторяющиеся элементы и их количество, т.е. для массива {5, 5, 5, 8, 8} будет 5: 3, 8:2
6. Вывести в обратном порядке члены массива (метод принимает аргументом массив)
7. Вывести индекс элемента массива который равен строке "this", если их несколько, то через запятую
8. Метод принимает массив чисел на вход и отдает на выход массив тех же чисел, но заменив отрицательные положительными. Нули не трогать.
9. Метод принимает массив на вход, и отдает новый массив, в котором каждый член это сумма этого числа и следующего. Последнее число останется таким какое есть.
10. Метод принимает массив на вход и отдает новый массив где первым элементом является сумма первого и последнего. Для второго элемента сумма второго и предпоследнего.
11. Метод принимает на вход массив чисел и отдает на выход массив вдвое больше. Первый и второй элемент нового массива равны первому элементу первого массива. Третий и четвертый элементы равны второму элементу первого массива. Типа {1, 2, 3}  $\rightarrow$  {1, 1, 2, 2, 3, 3}
12. Метод принимает на вход массив чисел и отдает отфильтрованный – т.е. без дубликатов. т.е. {1, 1, 2, 3}  $\rightarrow$  {1, 2, 3}
13. Метод принимает 2 массива чисел и выводит новый массив, где каждый член массива это произведение членов соответственно. т.е. {1, 2, 3};{4, 5, 6}  $\rightarrow$  {4, 10, 18}