

# Типы функций и переменные

Как одно работает со вторым

## Содержание

1. Передаем результат функции в аргумент другой функции
2. Переменные, создание и переименование

### 1. Передаем результат функции в аргумент другой функции

```
1 public class Main {
2
3     public static void main(String[] args) {
4         showSum(3, 4);
5         showSum(5, 6);
6         showDifference(1, 2);
7         showDifference(3, -4);
8     }
9
10    private static void showSum(int number1, int number2) {
11        print(number1 + number2);
12    }
13
14    private static void print(int number) {
15        System.out.println(number);
16    }
17
18    private static void showDifference(int number1, int number2) {
19        print(number1 - number2);
20    }
21
22 }
```

Main

Run: Main x

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

7  
11  
-1  
7

Process finished with exit code 0

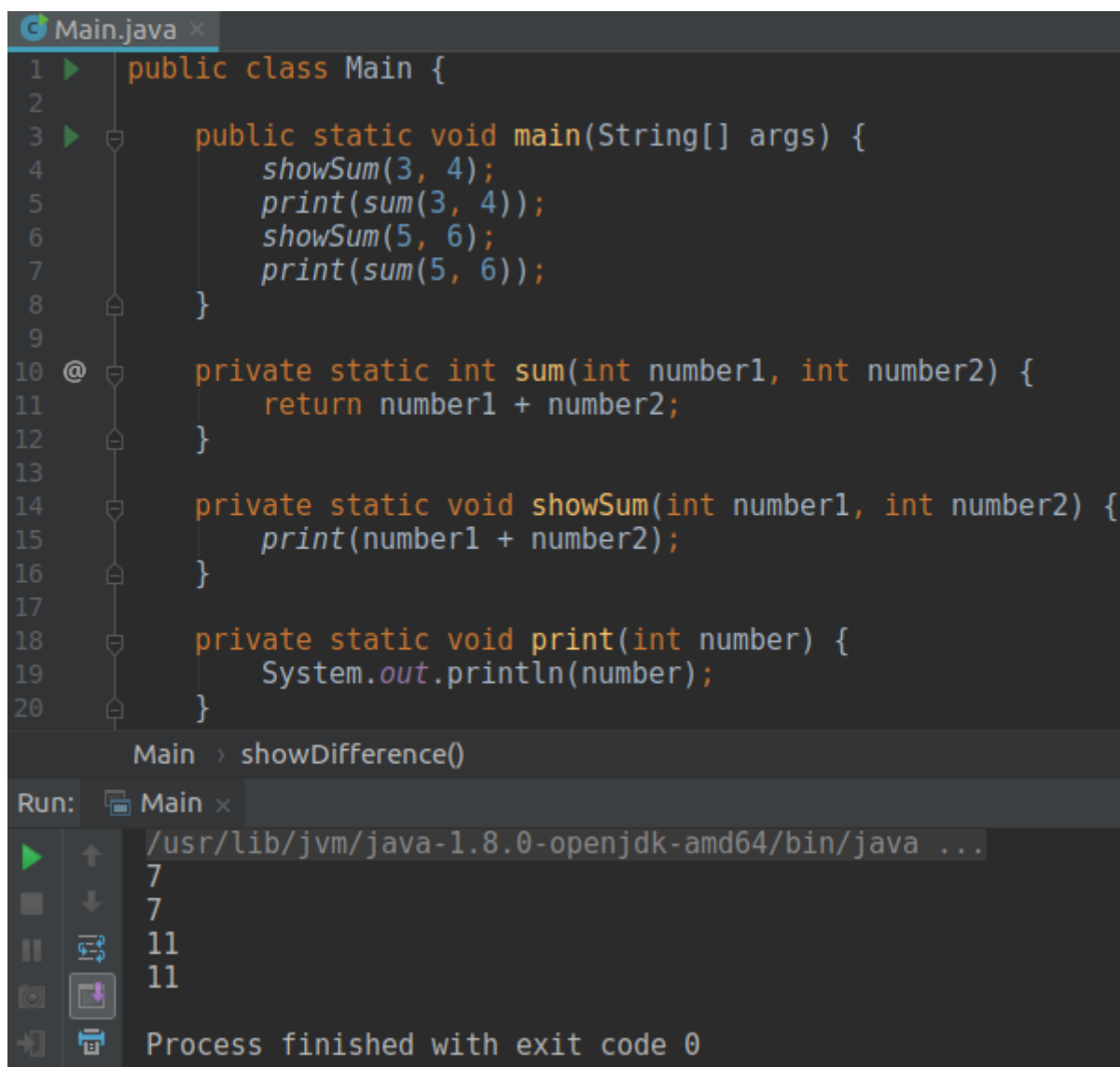
Начнем с того, что сделаем часть д/з из предыдущей лекции. Напишем метод, который покажет нам разницу 2 чисел на экран. Как видите, числа могут быть и отрицательными, именно поэтому 3 - (-4) будет 7. Все как и в школе, никакой магии. Но наши функции делают 1 действие и выводят результат на экран. А что если нам понадобится сначала произвести

одно действие, после второго и уже потом вывести на экран результат? Например рассмотрим такое математическое выражение

$$(a+b)*(c-d)$$

Согласитесь, мы не можем писать по 100 методов для всех случаев. Нам понадобится всего 3. Первый метод просто сложит 2 числа, второй метод найдет разницу 2 чисел и уже третий метод перемножит их. И уже потом можно вывести результат на экран. Вообще постарайтесь запомнить простое правило – метод должен делать лишь 1 действие. В нашем случае метод showSum делает как будто бы 1 действие, но на самом деле 2 - ведь сначала он складывает числа и уже потом показывает результат на экран. Не надо так. Давайте упростим.

Было бы классно, если бы метод смог принять 2 числа и вернуть результат, без показа его непосредственного результата на экран. Настало время вернуться к ключевому слову void. Оно дословно означает пустота. Т.е. метод у которого перед именем написано void (e.g. void showSum) не будет ничего возвращать в место вызова, или же скажем так возвращать пустоту. Что означает вернуть что-то в место вызова? Чуть ниже поймем. А сейчас давайте пока перепишем метод showSum так, чтобы он принимал 2 числа и возвращал сумму.



```
1 public class Main {
2
3     public static void main(String[] args) {
4         showSum(3, 4);
5         print(sum(3, 4));
6         showSum(5, 6);
7         print(sum(5, 6));
8     }
9
10    private static int sum(int number1, int number2) {
11        return number1 + number2;
12    }
13
14    private static void showSum(int number1, int number2) {
15        print(number1 + number2);
16    }
17
18    private static void print(int number) {
19        System.out.println(number);
20    }
21 }
```

Run: Main x

```
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
7
7
11
11
Process finished with exit code 0
```

Давайте посмотрим на код на линии 10. Мы объявили функцию `int sum(int number1, number2)`, посмотрите внимательно, теперь вместо `void` у нас `int`. Т.е. теперь вместо пустоты наш метод

вернет в место вызова число. И давайте посмотрим на код на линии 5. По сути он делает то же самое что и код на линии 4. Просто метод `showSum` сначала складывает 2 числа и передает результат в аргумент функции `print` внутри себя, а на линии 5 мы видим следующее

`print(sum(3,4))`

Вывести на экран сумму чисел 3 и 4. Но секунду, метод `print` принимает же аргументом число, так? Как мы передали туда функцию `sum`? Нет, это не так. Мы передали в аргумент функции `print` число, но оно рассчитывается в методе `sum` и возвращается в место вызова. т.е. на самом деле происходит следующее. Компилятор видит вызов метода `print` которому нужно передать число. Но он видит вызов метода `sum`, который принимает 2 числа и возвращает в место вызова сумму, тогда он сначала складывает числа и после уже отдает результат в метод `print`. По сути происходит следующее

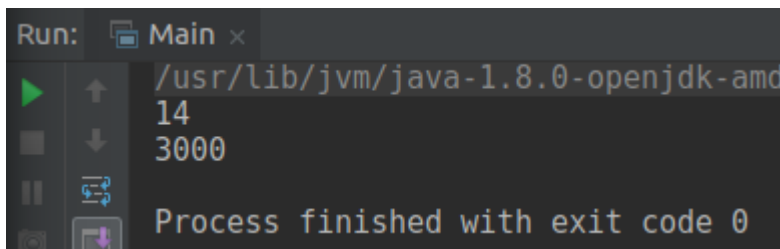
`print(sum(3,4))` → `sum(3,4)` → `print(7)`

Давайте также по линиям кода посмотрим как пойдет компилятор: 5,10,11,12, 18, 19, 20, 6...

Запомните, если ваш метод принимает аргумент, а вместо него есть вызов другой функции, то сначала компилятор должен выполнить вызов внутренней функции и только после уже первой. Надеюсь понятно, если нет, то просто продолжайте читать эту лекцию и повторять со мной. Вообще понимание иногда не приходит сразу, поэтому нужно дать время мозгам на усвоение материала.

Итак, вроде разобрались, можно написать метод, который вернет в место вызова некий результат. Давайте тогда допишем методы для разницы и умножения с возвращаемым типом числа. И да, надеюсь вы заметили ключевое слово `return` на линии 11. Это и значит – вернуть значение. Можете оставить эту лекцию на время пока сами не напишете код, после вернуться и проверить правильность.

```
1  ▶ public class Main {
2
3  ▶  public static void main(String[] args) {
4      print(multiply(sum(3, 4), difference(7, 5)));
5      print(multiply(sum(40, 60), difference(300, 270)));
6  }
7
8  @  private static int sum(int number1, int number2) {
9      return number1 + number2;
10 }
11
12 @  private static int difference(int number1, int number2) {
13     return number1 - number2;
14 }
15
16 @  private static int multiply(int number1, int number2) {
17     return number1 * number2;
18 }
19
20  private static void print(int number) {
21      System.out.println(number);
22  }
23 }
```



```
Run: Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd
14
3000
Process finished with exit code 0
```

По началу код на линии 4 может пугать, но ничего страшного в этом нет. Давайте разбираться. Мы пытаемся вывести на экран (print) произведение (multiply) чисел 3 и 4 (sum) и разницы чисел (difference) 7 и 5. т.е. математический результат вычисления  $(3+4)*(7-5)$ .

Что же произойдет с компилятором? Сначала он найдет сумму, после найдет разницу, далее найдет произведение и после уже передаст результат в метод print. Надеюсь ничего сложного здесь нет.

И кто-то может сказать: ок, здесь пример не такой вроде сложный, но что если нам понадобится посчитать что-то реально длинное, тогда у нас будет код на сто символов и не будет читаться? Было бы неплохо как в начальной школе расписать все действия. Ну помните?

1)  $3 + 4 = 7$

2)  $7 - 5 = 2$

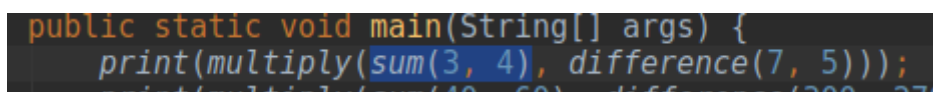
3)  $7 * 2 = 14$

Ответ: 14

Именно поэтому и в программировании есть такая замечательная штука как переменные;

## 2. Переменные, создание и переименование

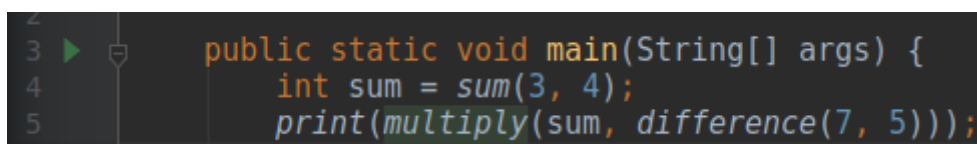
Итак, давайте наконец уж воспользуемся мощными преимуществами нашей среды разработки. На линии 3 выделите `sum(3, 4)` как выделяете текст



```
public static void main(String[] args) {
    print(multiply(sum(3, 4), difference(7, 5)));
    print(multiply(sum(40, 60), difference(200, 274)));
}
```

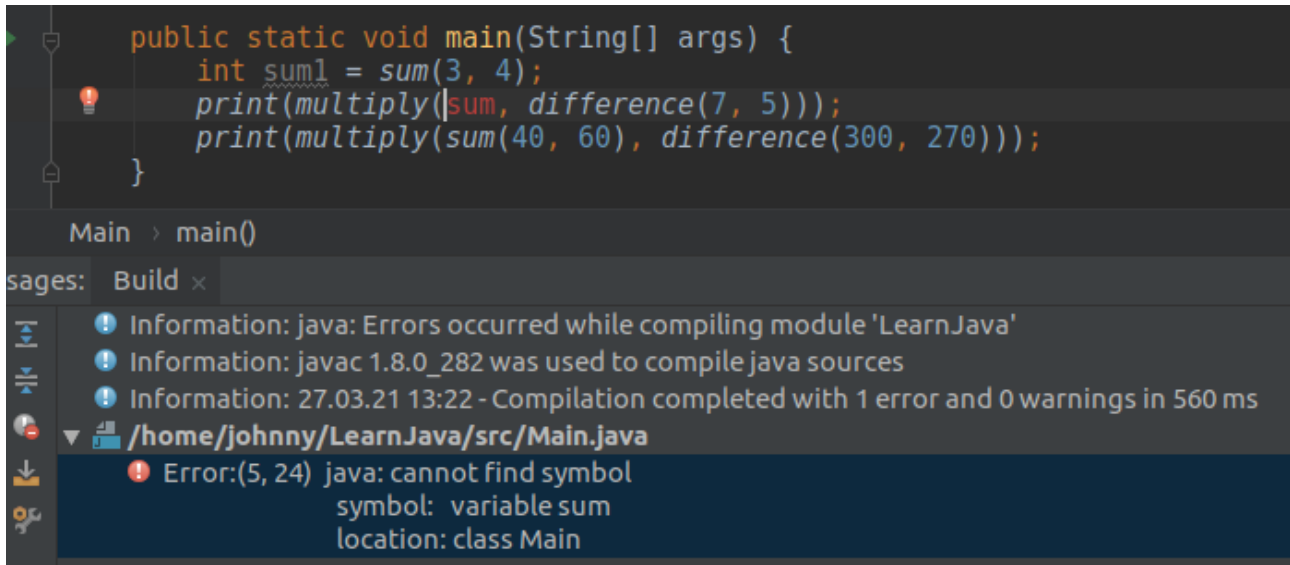
Нажимаем правый клик мышкой на выделенном участке, в открывшемся меню ищем Refactor, там уже ищем Extract и первая строка Variable. Или же есть горячая клавиша `Ctrl+Alt+V`.

И вуаля, мы получаем следующее: результат действия (суммы двух чисел) сохраняется в переменную (сразу можно дать имя этой переменной). Наша среда разработки умная и она видит, что результатом сложения чисел будет число и сразу же создает переменную правильного типа, т.е. `int`.



```
public static void main(String[] args) {
    int sum = sum(3, 4);
    print(multiply(sum, difference(7, 5)));
    print(multiply(sum(40, 60), difference(200, 274)));
}
```

Теперь у нас будет выполнено сложение чисел и сохранен результат в переменную с именем `sum`. Если вас смущает то, что у нас и имя функции `sum` и имя переменной `sum`, то можете дать переменной другое имя. Но не спешите просто набирать с клавиатуры другое имя. Как видите эта переменная используется уже на линии 5. И если вы поменяете имя переменной на линии 4, а на линии 5 не поменяете, то у вас сразу же среда разработки высветит ошибку и компилятор не исполнит программу. Как же тогда переименовать переменную правильно? Точно так же ставим курсор на переменную `sum`, нажимаем на правый клик, ищем Refactor → Rename или же горячая клавиша `Alt+Shift+F6`.



```
public static void main(String[] args) {  
    int sum1 = sum(3, 4);  
    print(multiply(sum, difference(7, 5)));  
    print(multiply(sum(40, 60), difference(300, 270)));  
}
```

Main > main()

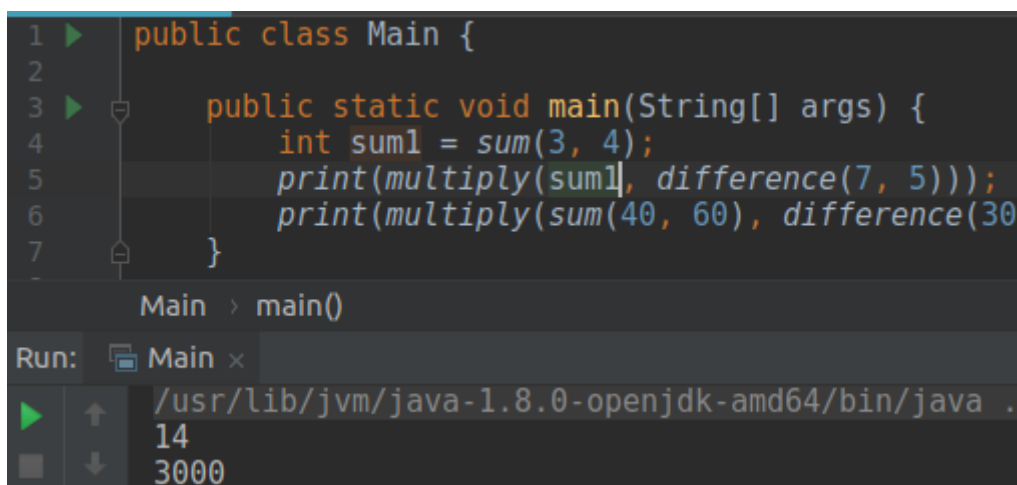
Build x

Information: java: Errors occurred while compiling module 'LearnJava'  
Information: javac 1.8.0\_282 was used to compile java sources  
Information: 27.03.21 13:22 - Compilation completed with 1 error and 0 warnings in 560 ms

/home/johnny/LearnJava/src/Main.java

Error:(5, 24) java: cannot find symbol  
symbol: variable sum  
location: class Main

Потому что компилятор ищет переменную с именем `sum` и не находит ее.



```
1 public class Main {  
2  
3     public static void main(String[] args) {  
4         int sum1 = sum(3, 4);  
5         print(multiply(sum1, difference(7, 5)));  
6         print(multiply(sum(40, 60), difference(300, 270)));  
7     }  
}
```

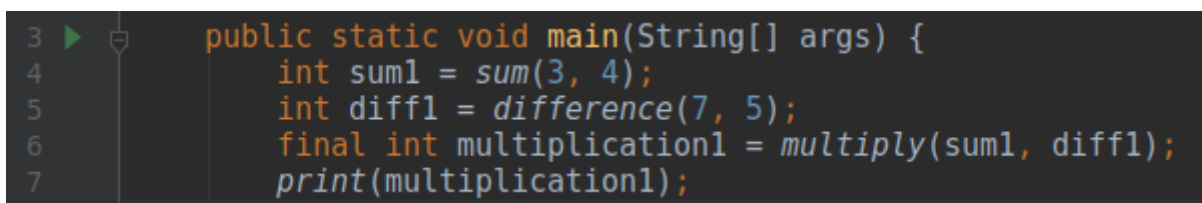
Main > main()

Run: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java .  
14  
3000

Так что постарайтесь сразу запомнить как правильно и безопасно менять имя переменной.

Теперь, можем ли мы так же поступить и для разницы? Давайте точно так же это сделаем. И сразу для умножения.



```
3 public static void main(String[] args) {  
4     int sum1 = sum(3, 4);  
5     int diff1 = difference(7, 5);  
6     final int multiplication1 = multiply(sum1, diff1);  
7     print(multiplication1);  
}
```

Теперь посмотрите на этот код и то, что было раньше. Стало проще, правда?

Линия 4: создаем переменную типа `int` с именем `sum1` и в ней храним (знак `=`) результат функции `sum(3, 4)`.

Линия 5: создаем переменную типа `int` с именем `diff1` и в ней храним результат функции `difference(7, 5)`.

Т.е. когда у вас компилятор достигает линии 4, то он выделяет в памяти (ваша ОЗУ) место под эту переменную `sum1`, после чего идет выполнять код, который суммирует числа и возвращает на линию 4 результат, который хранится в памяти. Но чтобы вы могли использовать потом этот результат вам нужно знать как обращаться к этой области памяти, потому и мы даем какие-то имена этим переменным. Сложно? Давайте проще поступим.

Представьте, что вы пишете отчет, в самом простом Microsoft Word. Вы начали его писать, но не закончили сразу и вам надо его сохранить на компьютере. Вы просто нажимаете сохранить файл и вуаля – у вас на рабочем столе незавершенный файл. После вы можете его открыть, отредактировать и отправить его на печать. А когда вы закончите работу с ним, то уже удалите его в корзину.

Вот вам еще одна аналогия. У вас есть шкаф в столе с 3 например ящиками. Вы берете и кладете в первый ящик какую-то вещь. Во второй ящик кладете второй предмет на следующий день. И на третий день вы кладете в третий ящик третий предмет. После уже на четвертый день вы достаете из первого ящика то, что положили ранее, из второго ящика второй предмет и третий из третьего ящика и что-то делаете с этими тремя предметами.

Та же история и с переменными. Нам нужно сложить числа, потом найти разницу других чисел, после найти произведение того, что мы сохранили в первой переменной и второй. И уже потом вывести на экран что получилось.

Кстати, вы наверно заметили что среда разработки при выделении переменной дает возможность не только дать ей имя, но и сделать ее `final`. Объясню это в следующих лекциях.

А пока вот вам домашнее задание. Помните что у нас было написано во второй лекции? Там были 2 строки. Я предлагаю вам написать следующий код:

метод который берет 2 строки и складывает их, между которыми добавляет символ переноса строки. Но не передавайте в мой метод эти строки, а передайте переменные, которые вы создали до этой линии.

т.е. в мейне у вас будет

объявление переменной строки 1

объявление переменной строки 2

объявление переменной строки3 и вызов функции которая складывает переменную1 и переменную 2

вызов метода принт с переменной строка3

Попробуйте по аналогии с числами, я уверен у вас получится, а если нет, то ничего страшного, мы начнем следующую лекцию именно с этого упражнения. Удачи!