

# Строки и числа

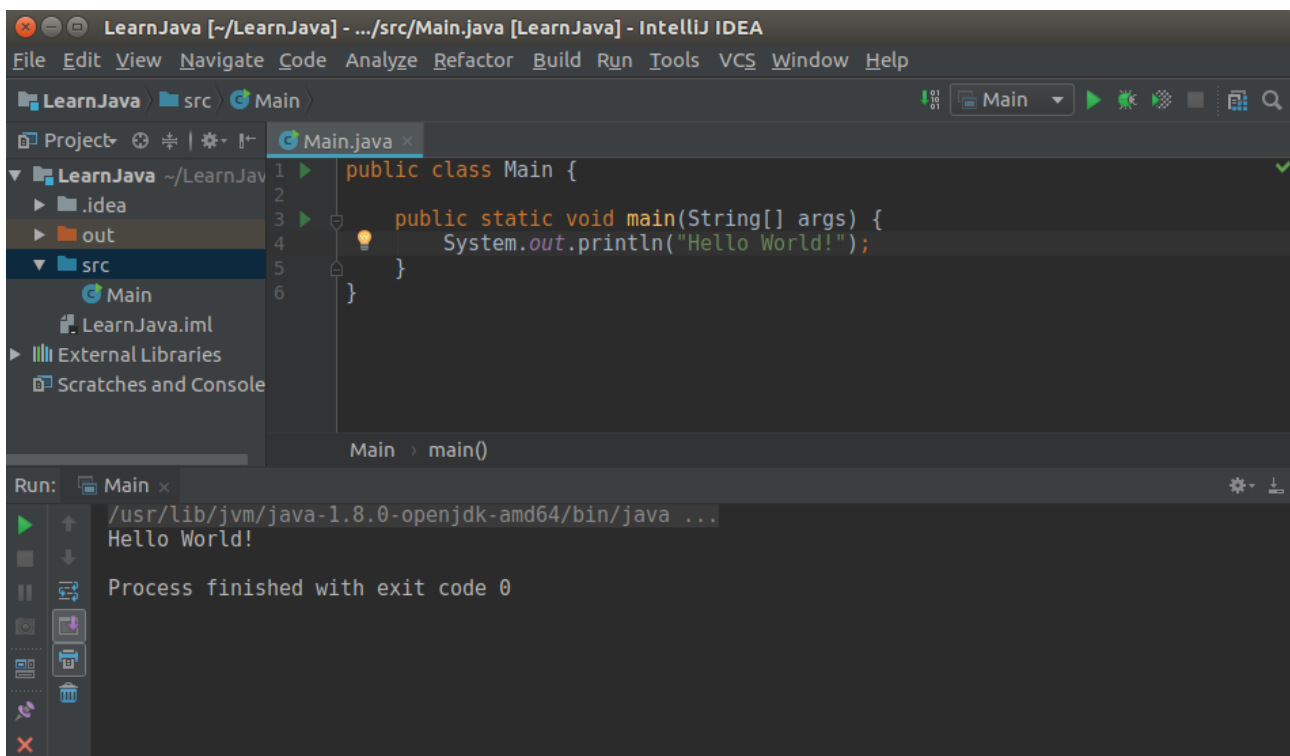
## Классы строк и чисел

### Содержание

1. Переходим от веб-сайта к продвинутой среде разработки
2. Складываем строки, введение в числа
3. Складываем числа

### 1. Переходим от веб-сайта к продвинутой среде разработки

Итак, друзья мои, если вы дошли до этой лекции и у вас все получилось, то я вас во-первых поздравляю и во-вторых я предполагаю что вы уже умеете гуглить. Если все же нет, то не страшно. В этой лекции нам будет удобнее использовать более подходящую среду разработки нежели веб-сайт, но если у вас возникнут трудности с установкой или же у вас нет времени на это, то ничего страшного, продолжайте учиться в вебе. По сути своей особой разницы нет, она лишь в удобстве и фишках – таких как подсказки среды разработки.

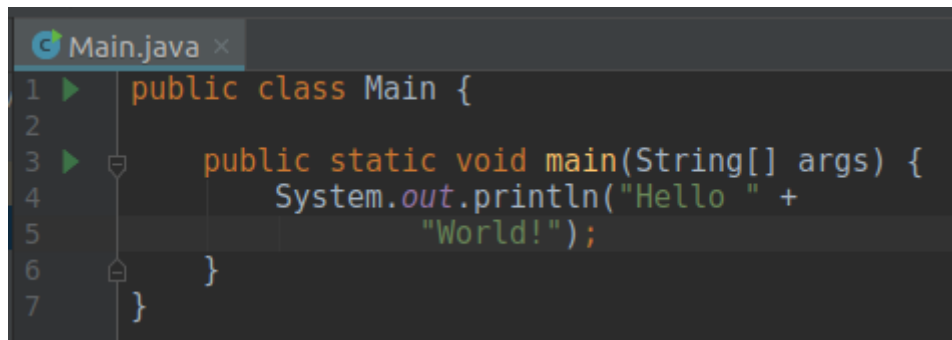


Итак, если вы готовы установить среду разработки (продвинутый текстовый редактор с компилятором джава), то я предлагаю вам самостоятельно это сделать. Для этого вбейте в гугл intellij idea, скачайте и установите бесплатную версию Community Edition. Далее создайте проект и в папке src создайте java class. Дальше вы знаете. Если вы попробуете и у вас ничего не получится, то ничего страшного. Попробуйте второй раз, попробуйте

разобраться. Однажды я услышал такое – если вы в состоянии установить среду разработки, то вы сможете стать программистом. В принципе довольно справедливо. И еще раз, если у вас пока сомнения и вы не уверены тратить время на установку среды разработки или нет, то не тратьте, просто пишите в браузере.

## 2. Складываем строки, введение в числа

Итак, имеем нашу программу Hello World! Давайте поставим курсор на втором слове и нажмем Enter и посмотрим что будет.

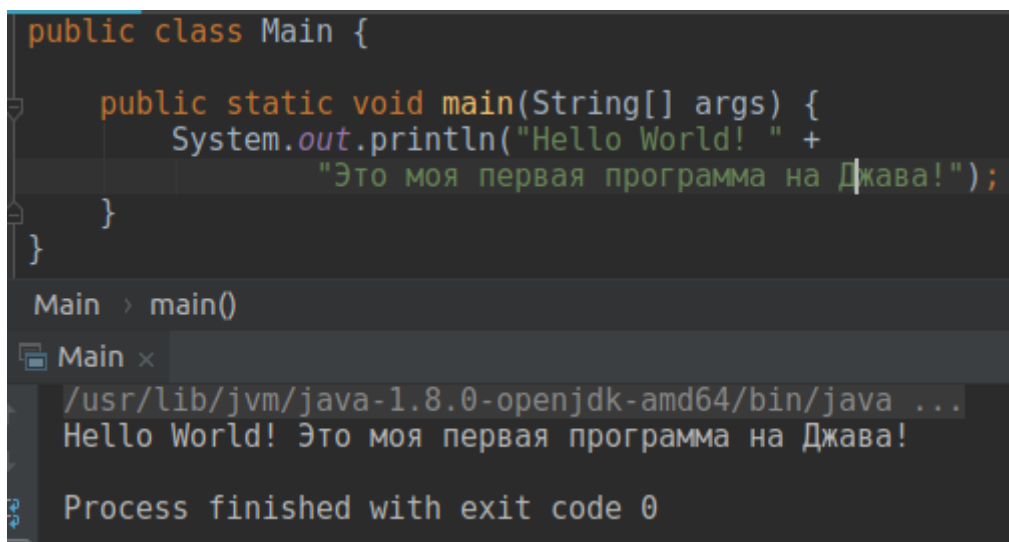


```
Main.java x
1 public class Main {
2
3     public static void main(String[] args) {
4         System.out.println('Hello ' +
5             'World!');
6     }
7 }
```

Если вы пишете код на веб-сайте и при нажатии на ввод у вас не получилось то, что у нас на скриншоте, то сделайте это сами.

Итак, что же произошло? Строка Hello World разделилась на 2 и соединяется знаком +. Чем же отличается этот код от того, что был на первом скриншоте? Ничем. Можете проверить нажав на Run. Но подождите, что это у нас получилось – мы складываем строки? Как числа?

Конечно же нет. Это называется конкатенация. Мы добавляем к существующей строке еще одну. Где же это могло быть полезно? А например в том случае, когда вы пишете очень длинную строку. Или же вы хотите визуально разделить 2 строки как в прошлом уроке. Давайте вспомним что было в предыдущей лекции.



```
public class Main {
    public static void main(String[] args) {
        System.out.println('Hello World! ' +
            'Это моя первая программа на Джава!');
    }
}

Main > main()
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
Hello World! Это моя первая программа на Джава!
Process finished with exit code 0
```

Но подождите, ведь в консоли это все тот же текст в 1 линию. Как же мы тогда можем и в консоли разделить их как в коде? Разве то, что я разделил их в коде не должно было повлиять на то, что я увижу в консоли? Оказывается нет. В коде мы можем разделить 1 строку на 5 или 500 штук по 1 букве, но это не значит что в консоли будет больше 1 линии. Знак + в коде лишь для удобства.

Но мы помним, что если написать

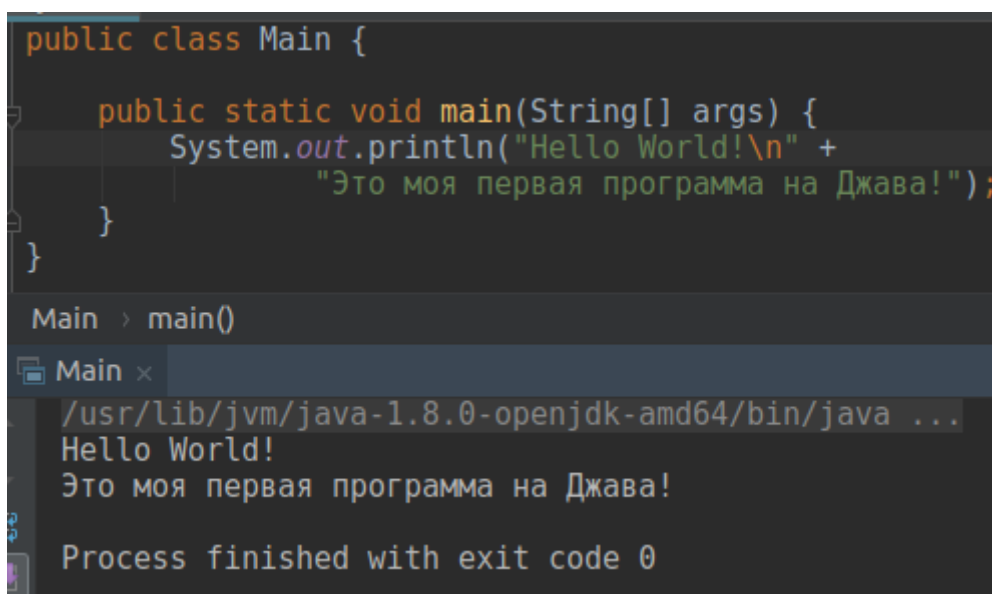
```
System.out.println("Hello World!");
```

```
System.out.println("|Это моя первая программа на Джава!");
```

то увидим в консоли 2 строки, одна под другой. Почему же так? Ведь и там и там 2 строки, строка же это любые символы между двумя двойными кавычками, не? Да, но дело в том, что функция системного класса `out.println` делает то, что выдает на экран строку в аргументе функции и... переносит курсор (каретку... ну фигню эту которая мерцает как в ворде) на следующую линию. И здесь было бы логично задать такой вопрос: ок, а можно как-нибудь не писать 2 раза `System.out.println`, а юзать его 1 раз и разделить как-нибудь строку на 2? Как будто если бы существовал символ для разделения строки. Вот именно! Такое существует.

`\n`

Давайте же попробуем!



```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello World!\n" +  
            "Это моя первая программа на Джава!");  
    }  
}
```

Main > main()

Main x

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

Hello World!

Это моя первая программа на Джава!

Process finished with exit code 0

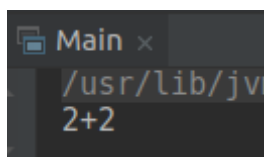
Как вы думаете, если мы вернем строку как было, без знака + между ними, но оставим обратный слеш и n, то вывод в консоль поменяется? Оставлю этот вопрос на самостоятельное изучение.

Целью этой лекции является ознакомление вас с таким понятием как числа. И чтобы продемонстрировать вам зачем они нужны мы попробуем сделать простые арифметические действия со строками.

Как вы думаете, что выведет на экран следующий код?

```
public static void main(String[] args) {  
    System.out.println("2+2");  
}
```

4? Может 5? Шутка. Не нужно гадать (хотя можно). Давайте просто запустим.



Main x

/usr/lib/jvm/

2+2

Неожиданно? Может нам нужно было сделать иначе? Например так?

```
public static void main(String[] args) {  
    System.out.println("2"+"2");  
}
```

Запустим? Я очень удивлюсь (и думаю вы тоже) результату.

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("2"+"2");  
    }  
}  
Main  
Main x  
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/j  
22  
Process finished with exit code 0
```

Мы говорили о том, что строка это все что угодно, если оно с двух сторон обрамлено двойными кавычками, так? А еще мы говорили что знак + между 2 строками всего лишь добавляет правую к левой. Т.е. что произошло у нас здесь? Компилятор видит строку 2 и выводит ее на экран. А после... добавляет к ней еще одну 2. И получается 22.

Я надеюсь что пока что вам все понятно и все идет логично. Мы же работаем со строками, а строки нельзя ни прибавить друг к другу, ни отнять, ни умножить – ничего. И здесь мы плавно переходим к такому понятию как числа. И несложно догадаться – если символ 2 обрамленный двойными кавычками является лишь строкой, а не числом, то скорее всего нужно просто напросто... удалить двойные кавычки, так? Пробуем?

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println(2+2);  
    }  
}  
Main > main()  
Main x  
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/j  
4  
Process finished with exit code 0
```

И здесь кто-то может спросить – а почему мы убрали двойные кавычки с обеих двоек? Недостаточно было бы убрать с одной? Ну что ж, нам нечего терять, давайте пробовать так.

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println(2+"2");  
    }  
}  
Main > main()  
Main x  
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/j  
22  
Process finished with exit code 0
```

Черт! Опять 22. Почему же так? И здесь мы впервые узнаем самую популярную фразу относительно языка Джава – это строготипизированный язык. Что это значит? А то, что мы четко разделяем понятия числа и строки. Все что обрамлено двойными кавычками это строка. С ней ничего нельзя сделать, максимум преобразовать в число, хотя бы попытаться. Мы посмотрим как это делать чуть позже. И все же, что тут происходит конкретно? Наш компилятор видит число 2 (кстати, вы заметили что наша среда разработки даже другим цветом выделяет число 2 и строку? Вот почему я рекомендую использовать IntelliJ Idea). И видит знак + и ожидает число, но вместо него видит строку и... сначала выводит число 2 и после уже строку 2. То же самое будет если поменять местами число 2 и строку 2. Попробуйте сами, если не верите на слово. И да, никогда не верьте на слово, как только вы это сделаете, вы перестанете быть программистом! Магия исчезнет!

Итак, что же мы поняли? Что нужно производить операции с один и тем же типом данных.

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println(Integer.valueOf("2") + 2);  
    }  
}  
Main > main()  
Main x  
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...  
4  
Process finished with exit code 0
```

Как и обещал: метод, который преобразовывает строку в число. Как мы видим, этот метод относится к некоему классу Integer. И скажу вам наперед – если вы вместо числа между двойными кавычками скормите ему пустую строку или букву или слово, то у вас ничего не получится. Но об этом будем говорить в других лекциях. Сейчас пока просто знайте как это делать. Для самых умных из вас у меня такой вопрос – у нас же есть класс для строк String, значит и у него мог бы быть метод, преобразующий число в строку, так? А есть ли такое и зачем оно – подумайте сами (а лучше попробуйте на практике).

### 3. Складываем числа

В предыдущей лекции мы узнали об истинном предназначении функций. Давайте представим, что нам нужно в нашей программе складывать 2 числа вместе и выводить их в консоль. Тогда бы нам нужно было бы писать длинное `System.out.println`, но мы уже научились писать собственные функции, которые упростят нам жизнь. В прошлый раз мы написали функцию, которая принимала бы аргументом 1 строку и ее просто выдавала бы на экран. А можем ли мы сейчас написать функцию, которая бы принимала аргументом 2 числа, и выводила их сумму на экран? Ведь было бы странно, если бы функции принимали лишь 1 аргумент на вход, не так ли? И если для строки мы пишем тип `String`, то для числа, как мы уже знаем, есть класс `Integer`. Ну что ж, пробуем? (Вы можете попробовать сами и после уже свериться с лекцией)

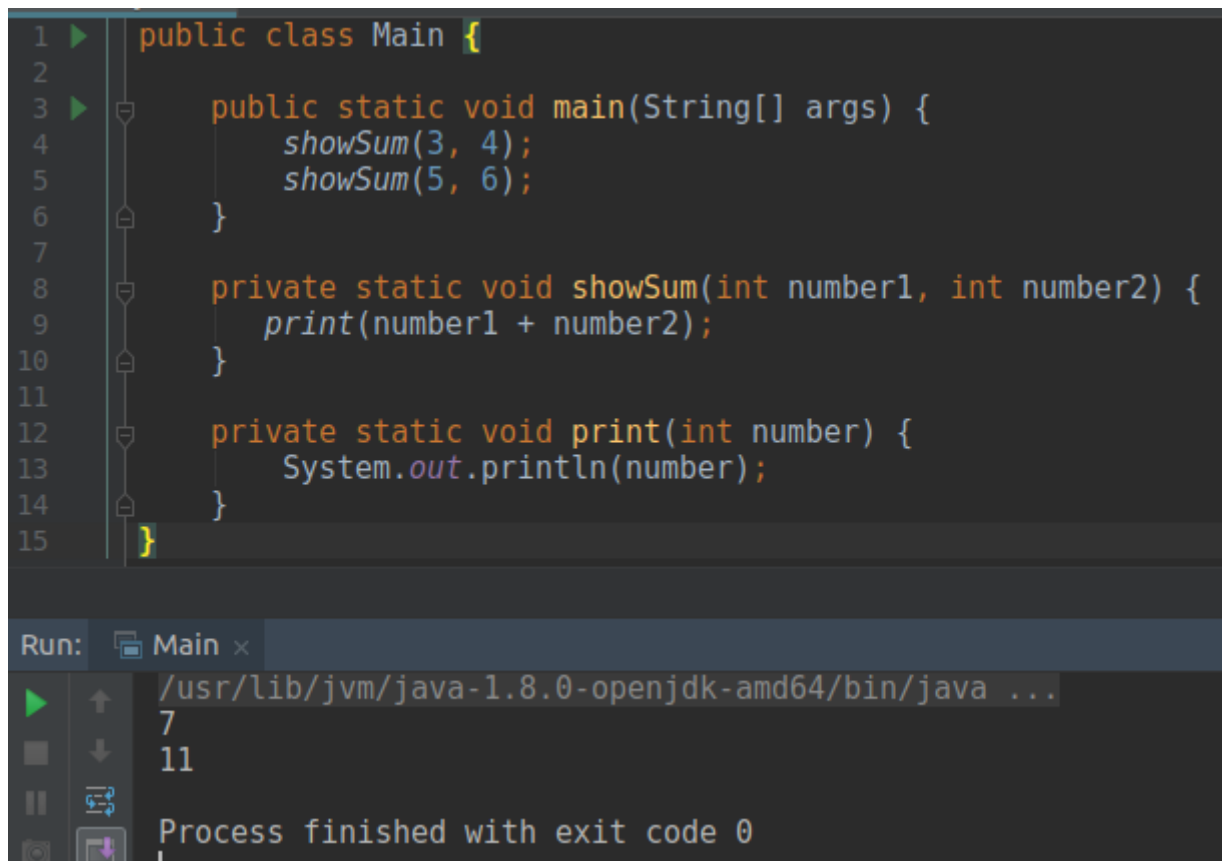
```
public class Main {  
    public static void main(String[] args) {  
        showSum(3, 4);  
        showSum(5, 6);  
    }  
    private static void showSum(Integer number1, Integer number2) {  
        System.out.println(number1 + number2);  
    }  
}  
Main > showSum()  
Main x  
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...  
7  
11  
  
Process finished with exit code 0
```

Обсудим подробнее код метода `showSum`. Во-первых вам нужно знать еще одно правило код-стайла Джава – принято писать имена методов нижним `camelCase`. Это когда первое пишем строчной буквой, а второе слово начинается с заглавной буквы, как горб у верблюда (отсюда и название). Далее идут 2 аргумента – `Integer number1`, `Integer number2`. Да, как можно было догадаться, если у вас метод принимает более 1 аргумента, то их нужно разделить запятыми. Что касается чисел в названиях аргументов – то да, их можно использовать, но не начинать ими имя аргумента, т.е. не надо делать так – `1number` или `2Number`. Это не комильфо и угадайте что произойдет (а лучше попробуйте).

А чтобы нам проверить правильность написанной функции `showSum` нужно вызвать ее в методе мейн. Возьмем 2 простых числа и сложим их. И да, порядок аргументов в Джава важен, т.е. когда вы вызываете метод `showSum` с аргументами 3 и 4, то компилятор присваивает значение 3 аргументу `number1`, а 4 аргументу `number2`. Подумайте как это можно проверить и проверьте это.

И здесь я задам вам наверно обескураживающий вопрос, а вам не кажется, что для такой простой вещи как число (заметьте, целочисленное, в следующей лекции мы это проверим)

нам нужен в Джава целый класс. Может есть нечто попроще? Ответ – конечно же есть. Это же просто числа, зачем нам целый класс под это дело.



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

Run: Main x

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

Что же я поменял? Вместо класса Integer я написал int. Как видим, среда разработки выделила это слово другим цветом, таким же как и private static void, что означает зарезервированное слово в языке Джава (кстати да, вы не можете использовать int в качестве имени аргумента, т.е. нельзя написать int int, компилятор сойдет с ума, ибо он ждет после типа аргумента его имя, а не еще раз тип). И да, ключевое слово private мы обсудим попозже, вы можете его не писать совсем. Но секунду, если для строки у нас есть класс String, то почему мы можем использовать какое-то int вместо Integer? Пока что мы не будем вдаваться в подробности, я обещаю вам рассказать об этом позже, ок? А пока запомните, что можно и так и так использовать. И запомните пока, что int означает примитив. И это логично с одной стороны. Ведь что можно делать с целыми числами? Не особо много чего, да? Примитивные арифметические операции, такие как сложение, вычитание, умножение и деление. Для этих целей достаточно операторов как в математике: +, -, \*, /. А со строками можно делать много чего интересного, именно поэтому там не могло быть примитива. Проверьте, напишите string и вы увидите что будет.

Кстати да, я использовал наш метод из предыдущей лекции, только в нем поменял тип аргумента. Речь о коде метода на линии 12. Итак, еще раз разберем весь код. Компилятор ищет метод мейн. В нем видит вызов функции showSum. Находит его и присваивает аргументам значения. После чего ищет метод print(int number) и когда находит его на строке 12, то передает уже в него сумму аргументов number1 и number2, т.е. 7. Что значит, что вызывается в итоге метод print с аргументом number равным 7. И после чего компилятор возвращается на линию 5 и все повторяется еще раз. т.е. порядок выполнения такой:

3,4,8,9,12, 13, 14,10, 5,8,9,12,13,14,10,6. Мы сможем это проверить (когда-нибудь, но не сейчас) в будущих лекциях.

Подведем итог этой лекции – вы узнали, что для строки есть класс `String`, а для чисел есть `Integer`. В котором есть метод преобразования строки в число. К числам можно применить простые арифметические операторы (кстати, вот вам домашнее задание, напишите методы по аналогии с суммой чисел для остальных действий – вычитания, умножения и деления), а строки можно добавлять друг к другу. Так же вы узнали, что кроме класса `Integer` можно использовать примитив `int`. И вы должны понимать, что все возможные методы над числами можно найти в классе `Integer`. А `int` лишь для объявления типа числа в аргументах (и не только и об этом будет следующая лекция).

Надеюсь что эта лекция не была сложной для понимания несмотря на то, что она получилась немного длиннее чем предыдущие. Но привыкайте понемногу к нарастанию материалов. Я буду стараться делать лекции короткими, простыми и понятными, но это зависит от темы конечно же. Удачи!