

## 7.6

### Символы

**00:00–00:31**

#### **Введение**

Привет! В этом уроке мы поговорим ещё об одном примитиве — `char`. `Char` — это символ, и его можно задавать в одинарных кавычках, в отличие от строк, которые задаются двойными кавычками. Кроме этого, `char` можно задавать числом, поскольку переменная типа `char` хранится именно в виде числа. Это число представляет собой номер символа в некой таблице, называемой кодировкой. Кодировки бывают разные, и о них мы поговорим немного позже.

**00:31–01:10**

#### **Символ `char`**

В памяти символ `char` занимает 16 бит, то есть два байта. Несложно подсчитать максимально возможное количество символов, которые могут быть закодированы такими числами — их будет  $2^{16}$  в 16 степени, то есть 65 536.

Также номер символа бывает удобно задавать в виде числа в шестнадцатеричной системе:

- сначала пишется 0;
- потом буквы — X;
- далее число — номер этого символа в кодировке в шестнадцатеричной системе.

Вспомним, что в шестнадцатеричной системе используется 16 символов вместо 10, то есть цифры от 0 до 9 и буквы от A до F.

**01:10–03:39**

#### **Специальные символы**

Существуют специальные символы, которые задаются особым образом. Например, символ переноса строки в операционных системах Linux и Mac OS задается с помощью обратного слеша (\) и буквы n, n означает newLine, то есть новая строка.

```
char newLine = '\n';
```

Кстати, в операционной системе Windows перенос строки — уже два символа: обратный слеш и r, а также обратный слеш и n, и это уже строка, а не отдельный символ.

```
String nlWin = "\r\n";
```

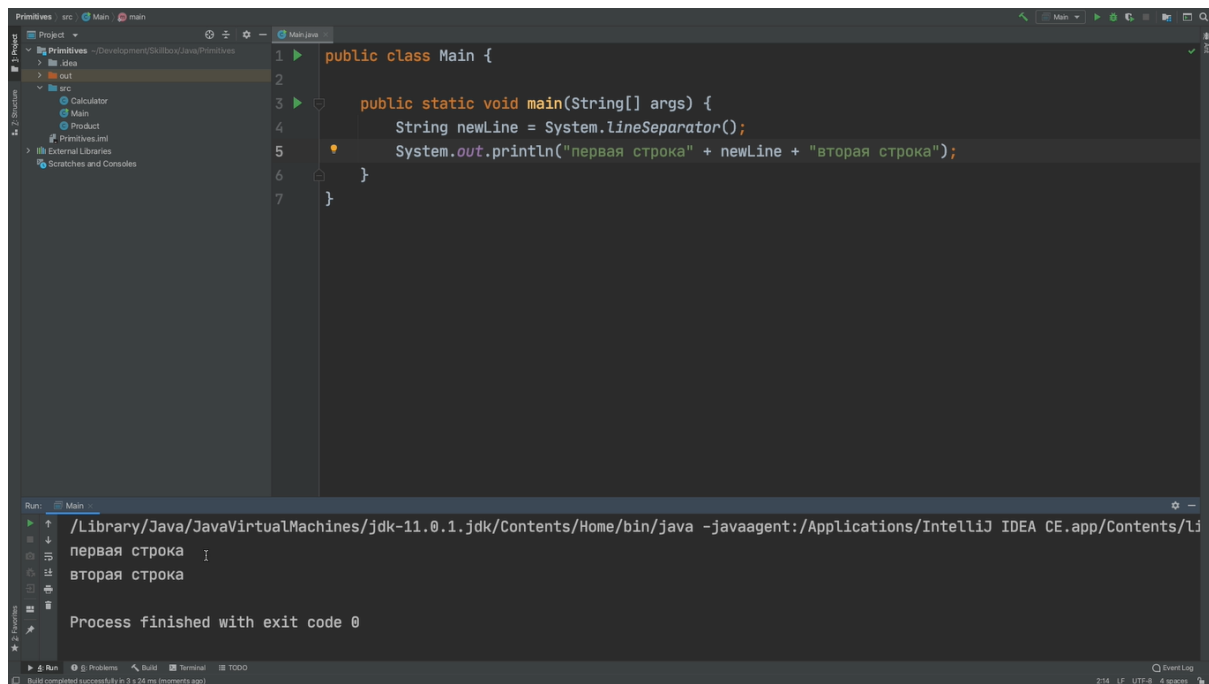
Такое обозначение переноса строки пришло к нам из времён печатных машинок: обратный слеш и r означают возврат каретки, а обратный слеш и n — перевод листа на одну строчку вниз.

Если вы хотите написать универсальный код, который будет совместим со всеми операционными системами, то для получения символа переноса строки используйте статический метод line separator класса System.

```
String newLineUniversal = System.lineSeparator();
```

Давайте посмотрим, как это выглядит в коде.

Создаём строку. Поскольку в Windows перенос строки — это два символа, создаём класс System, метод line separator и теперь выводим его в консоль, указывая System.out.println(“первая строка”+newline + “вторая строка”). Запускаем.



The screenshot shows the IntelliJ IDEA IDE. The main editor displays a Java file named `Main.java` with the following code:

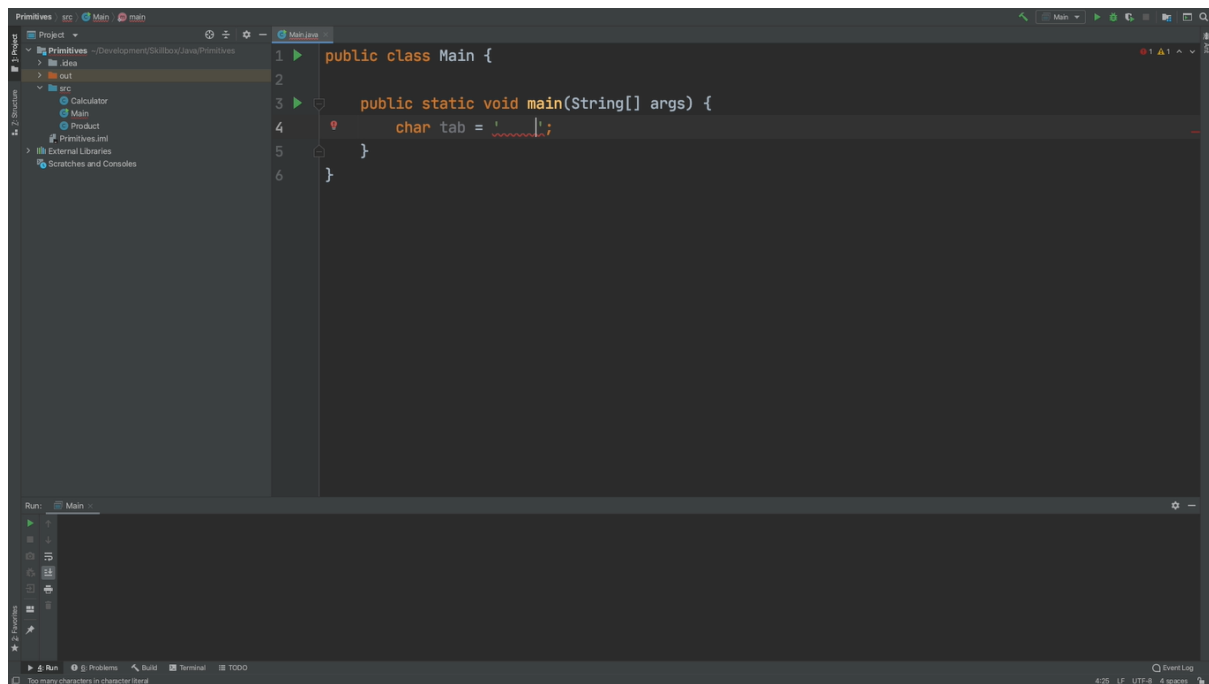
```
1 public class Main {  
2  
3     public static void main(String[] args) {  
4         String newLine = System.LineSeparator();  
5         System.out.println("первая строка" + newLine + "вторая строка");  
6     }  
7 }
```

The Run tool window at the bottom shows the execution output:

```
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib  
первая строка  
вторая строка  
  
Process finished with exit code 0
```

Вне зависимости от операционной системы и ввода команды переноса строки вывод в консоли ничем не будет отличаться.

Ещё один специальный символ — это символ табуляции. Он задаётся как обратный слеш (`\`) и `t`. Его, в отличие от символа переноса строки, можно ввести с клавиатуры нажатием клавиши табуляции, но так лучше не делать, поскольку в некоторых средах разработки при нажатии на клавишу табуляции будут вводиться пробелы.

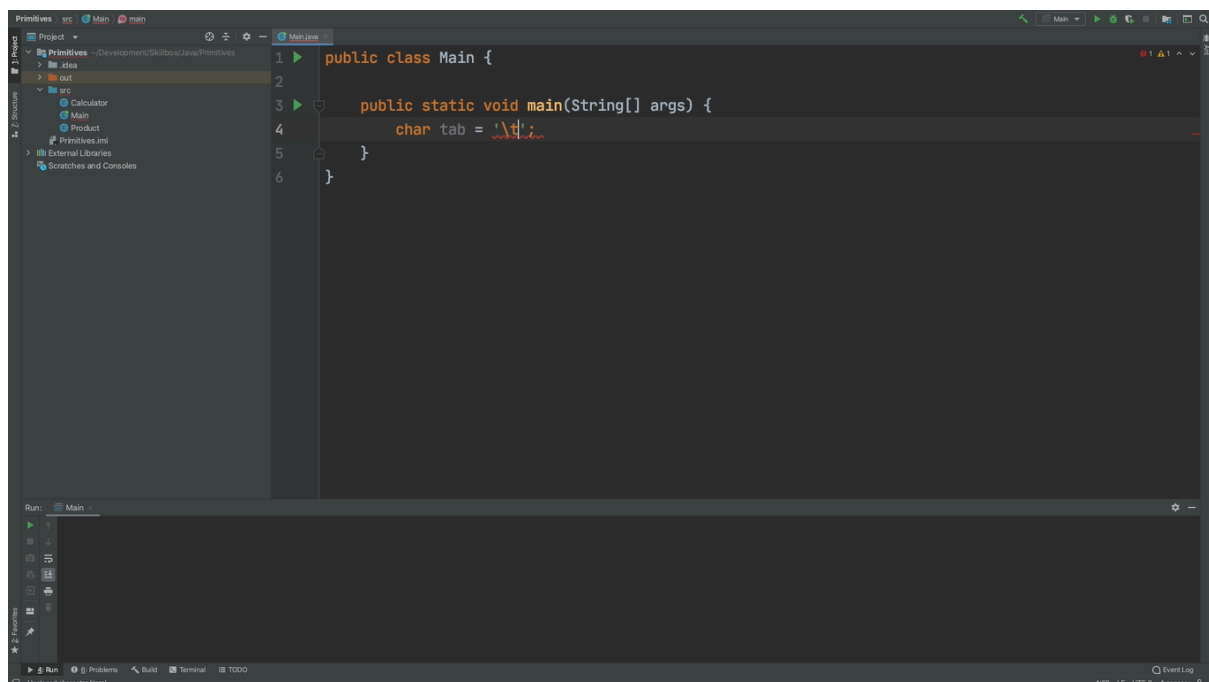


```
1 public class Main {
2
3     public static void main(String[] args) {
4         char tab = '\t';
5     }
6 }
```

The screenshot shows an IDE window with a project named 'Primitives'. The 'src' folder contains 'Main.java'. The code in 'Main.java' is as follows:

```
1 public class Main {
2
3     public static void main(String[] args) {
4         char tab = '\t';
5     }
6 }
```

The IDE status bar at the bottom indicates 'Run: Main' and '4:25 LF UTF-8 4 spaces'.



```
1 public class Main {
2
3     public static void main(String[] args) {
4         char tab = '\t';
5     }
6 }
```

The screenshot shows the same IDE window as above, but with a different status bar indicating 'Run: Main' and '4:23 LF UTF-8 4 spaces'.

Даже если у вас с клавиатуры символ табуляции вводится именно как символ табуляции, всё же лучше так не делать, потому что такой код будет недостаточно читаемым. Программный код всегда должен быть написан понятно, чтобы понимали его не только вы, но и те, кто работает с вами в команде.

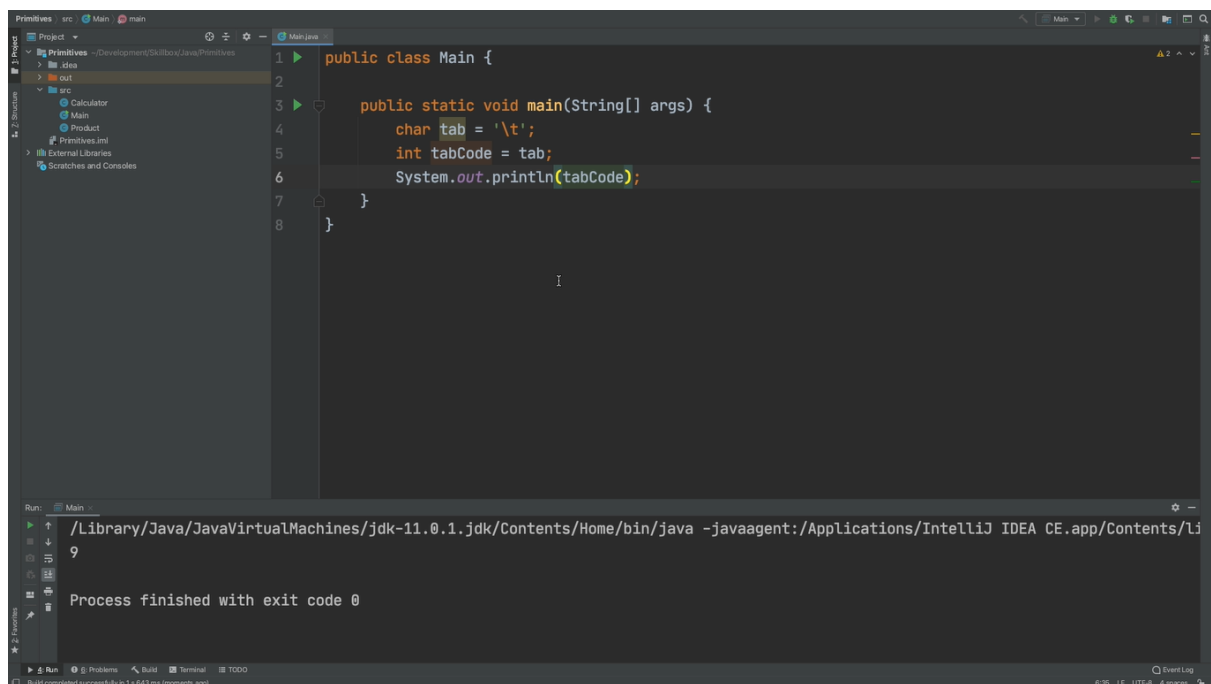
**03:39–06:09**

**Преобразование символов в код**

Давайте теперь посмотрим, как можно преобразовывать символы в коды, то есть в числа, и обратно. Давайте возьмём пример с табуляцией и посмотрим код этого символа.

Создадим число и присвоим ему переменную `tab`, она автоматически будет преобразована в число: `int tabCode = tab`.

Далее выведем в консоль код этого символа: `System.out.println("tabCode")`.



The screenshot shows an IDE window with a Java file named `Main.java`. The code is as follows:

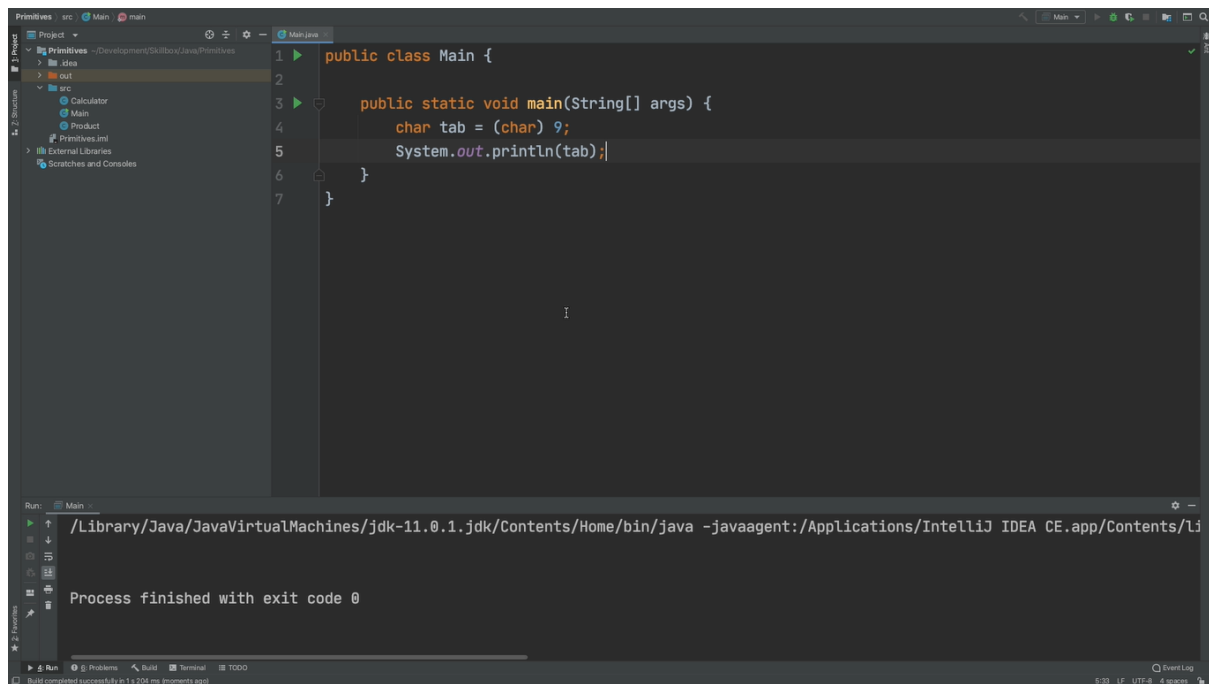
```
1 public class Main {  
2  
3     public static void main(String[] args) {  
4         char tab = '\t';  
5         int tabCode = tab;  
6         System.out.println(tabCode);  
7     }  
8 }
```

Below the code editor, the Run window shows the command executed: `/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li`. The output is `9`, which is the ASCII code for the tab character. At the bottom, it says "Process finished with exit code 0".

Вы видите, что код этого символа — 9.

Что касается обратного преобразования из числа в символ, то его можно осуществить, используя так называемое приведение типов.

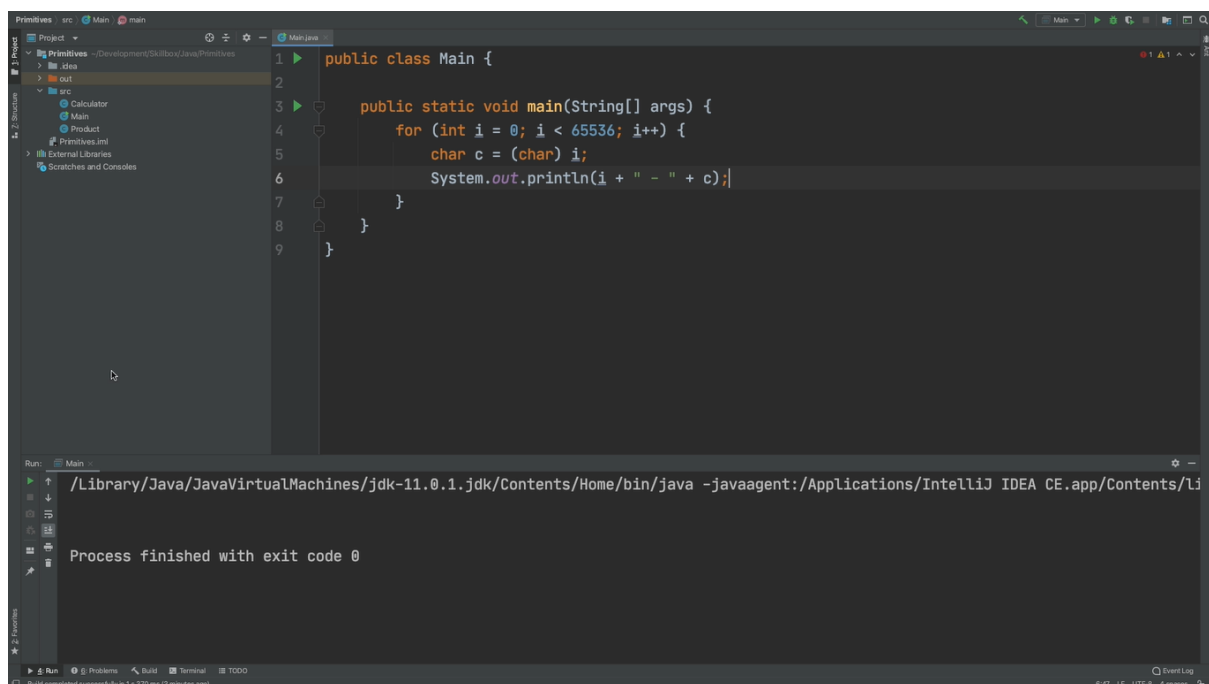
Для этого в первой строке нужно поставить круглые скобки вместо `'\t'`, написать тот тип, к которому мы приводим число, и само число, которое мы к этому типу приводим.



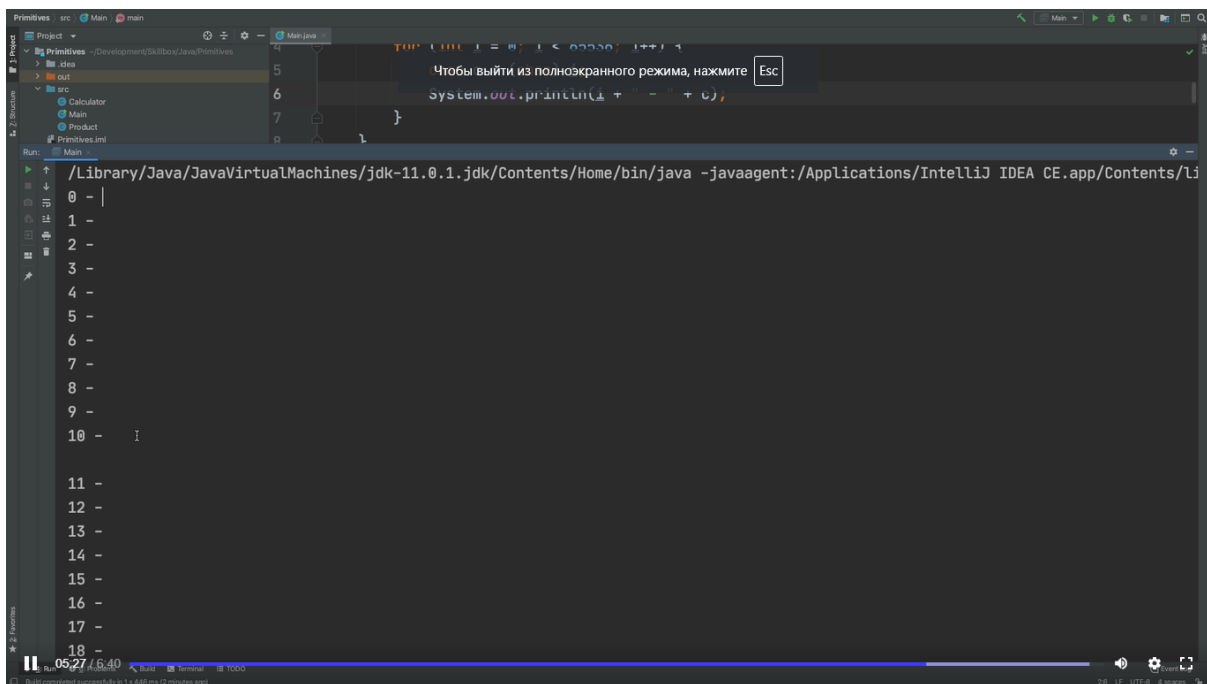
Здесь не видно символа, но он есть. :)

Давайте таким способом выведем в консоль символы, соответствующие всем числам, которые можно присвоить переменной типа char.

Для этого напишем цикл for и переберём числа от 0 до 65 536, преобразуем их в символы: сначала номер символа, потом дефис и потом сам символ.

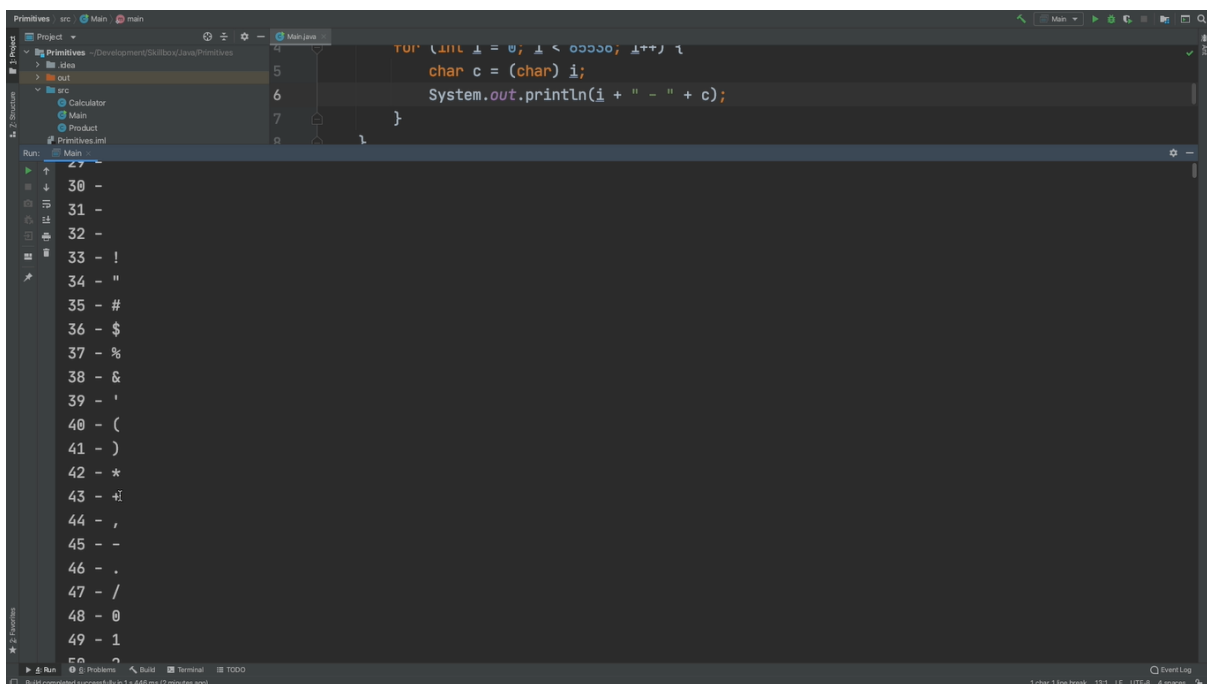


Давайте посмотрим, что вывелось в консоль.



```
Project: src | Main | main
Primitives - Development/Skillbox/Java/Primitives
  out
  src
    Calculator
    Main
    Product
  Primitives.iml

Run: Main
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
0 -
1 -
2 -
3 -
4 -
5 -
6 -
7 -
8 -
9 -
10 -
11 -
12 -
13 -
14 -
15 -
16 -
17 -
18 -
```



```
Project: src | Main | main
Primitives - Development/Skillbox/Java/Primitives
  out
  src
    Calculator
    Main
    Product
  Primitives.iml

Run: Main
27 -
30 -
31 -
32 -
33 - !
34 - "
35 - #
36 - $
37 - %
38 - &
39 - '
40 - (
41 - )
42 - *
43 - +
44 - ,
45 - -
46 - .
47 - /
48 - 0
49 - 1
```

06:08–06:40

## Итоги

Итак, в этом видео вы познакомились с примитивом `char`, узнали, как можно задавать переменные этого типа, узнали, что символы

кодируются числами, и познакомились с тем, как можно преобразовывать числа в символы и обратно.

На этом мы заканчиваем рассмотрение примитивов и в следующем видео поговорим о специальных классах-обёртках, которые дополняют все примитивы, а также о том, как преобразовывать примитивы в объекты таких классов и обратно.