

Еще 3 типа переменных

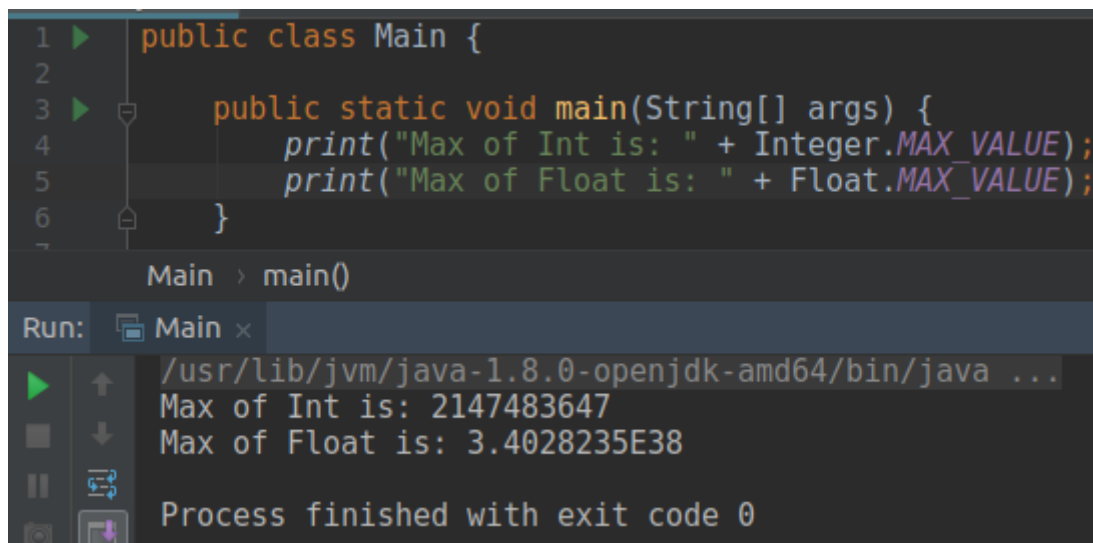
Преобразование одного типа в другой

Содержание

1. Еще 3 типа для чисел
2. Преобразование одного типа в другой

1. Еще 3 типа чисел

Давайте посмотрим внимательно на максимальные значения `int` и `float`.



```
1 public class Main {
2
3     public static void main(String[] args) {
4         print("Max of Int is: " + Integer.MAX_VALUE);
5         print("Max of Float is: " + Float.MAX_VALUE);
6     }
7 }

Main > main()

Run: Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
Max of Int is: 2147483647
Max of Float is: 3.4028235E38

Process finished with exit code 0
```

Вам не кажется, что разница между ними огромная? Сами посудите, 2 миллиарда с лишним и $3.4 \cdot 10^{38}$. Да, если вы не знали, то E38 означает 38 нулей. Это же огромное число. Зачем же мне использовать этот `float` если мне нужно сохранить всего лишь 3 миллиарда? И ответ на этот вопрос тоже есть. Представляю вашему вниманию еще один тип чисел – `Long`. Точно так же как и с `float`, это примитив, у которого есть класс.

А вообще мы бы могли легко проверить это на деле. Давайте создадим `int` переменную и попробуем в нее сохранить 3 миллиарда.

Как видим среда разработки подчеркнула наше число и выход из этой ситуации просто поменять `int` на `long`. Но как видим что-то пошло не так и ошибка осталась. Почему же? Дело все в том же, ведь как помните, для `float` мы добавляли букву `f` в конце чтобы различать тип `float` от типа `double`. Точно так же и с `long`. Нам нужно различать `int` и `long` и логично предположить что нужно написать букву `l` в конце. Попробуйте. И вы узнаете, что оказывается нужна заглавная буква `L`. Почему же так? А все потому что строчная `l` очень похожа на цифру 1 и чтобы не путать их, решено было использовать заглавную.

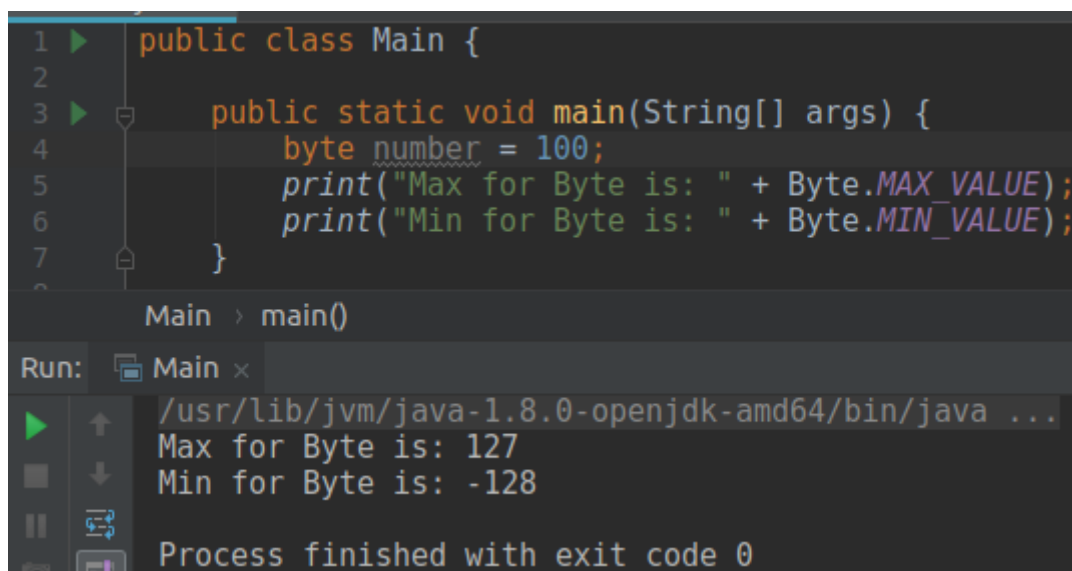
```
1 public class Main {
2
3     public static void main(String[] args) {
4         int number = 30000000000;
5     }
6
7     private static void print(String text) {
8         System.out.println(text);
9     }
10 }
Main > main()
Messages: Build x
Information: java: Errors occurred while compiling module 'LearnJava'
Information: javac 1.8.0_282 was used to compile java sources
Information: 30.03.21 12:19 - Compilation completed with 1 error and 0 warnings
/home/johnny/LearnJava/src/Main.java
Error:(4, 22) java: integer number too large: 30000000000
```

```
1 public class Main {
2
3     public static void main(String[] args) {
4         long number = 30000000000;
5     }
6 }
Main > main()
Messages: Build x
Information: java: Errors occurred while compiling module 'LearnJava'
Information: javac 1.8.0_282 was used to compile java sources
Information: 30.03.21 12:20 - Compilation completed with 1 error and 0 warnings
/home/johnny/LearnJava/src/Main.java
Error:(4, 23) java: integer number too large: 30000000000
```

```
1 public class Main {
2
3     public static void main(String[] args) {
4         long number = 30000000000L;
5         print("Max for Long is: " + Long.MAX_VALUE);
6         print("Min for Long is: " + Long.MIN_VALUE);
7     }
8 }
Main > main()
Run: Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
Max for Long is: 9223372036854775807
Min for Long is: -9223372036854775808
Process finished with exit code 0
```

Ну и давайте глянем какие минимальные и максимальные значения для них.

Но мы посмотрели с одной стороны. Если нам нужно число больше чем 2 миллиарда то мы не можем использовать `int`. А что если нам нужна переменная для маленьких чисел? Типа меньше 100. Зачем использовать `int`? Представьте, что вы кладете в ящик стола маленькую флешку или пуговицу. И ничего кроме нее вы не можете положить туда. Согласитесь, нерациональное использование хранилища. И кто-то подумает, ну тогда нам нужно хранилище поменьше для таких вещей и будет совершенно прав. В языке Java специально для маленьких чисел есть тип `byte`.



```
1 public class Main {
2
3     public static void main(String[] args) {
4         byte number = 100;
5         print("Max for Byte is: " + Byte.MAX_VALUE);
6         print("Min for Byte is: " + Byte.MIN_VALUE);
7     }
8 }
9
10 Main > main()
Run: Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
Max for Byte is: 127
Min for Byte is: -128
Process finished with exit code 0
```

Так что друзья мои, если вы понимаете, что вам нужно хранить максимум число 127, то вам лучше взять тип `byte`. Ведь он в памяти занимает куда меньше места, чем тот же `int`.

Но давайте поговорим про настоящее использование типа `byte` в жизни. Рано или поздно вы столкнетесь с такой задачей – прочитать или записать данные. Представьте, что у вас ванна наполнена водой и вам нужно ее оттуда перелить в раковину например. Естественно, сделать это сразу не получится. И вы конечно же берете какой-нибудь маленький ковшик и с его помощью переливаете воду из ванны в раковину. Хотя согласен, вылить в раковину это больше похоже на удаление, чем перемещение, но все же. Итак, вы берете ковшик (`byte`) и опустошаете огромный кусок информации. С одной стороны вы делаете множество подходов пока не закончите, но с другой стороны вы используете очень мало ресурсов. 1 байт занимает в памяти очень мало места. Можете сами погуглить кстати.

И опять же встает вопрос, секунду, а что насчет чисел чуть больше 127, но меньше 2 миллиардов? Неужели ничего нет. Конечно же есть. `Short`. Но скажу вам честно, я очень редко видел использование этого типа данных. Если `byte` используется для перекачки данных из одного места в другое, а `float` часто используется в Android (ну как часто, иногда), то использование `short` я наверно не вспомню за жизнь. Потому что зачастую используются `int` и `double`. И да, `long` для дат в Android. `Int` для любых чисел целого типа и `double` для всех чисел с десятичной частью. Но зачем вам знать про эти типы? Возможно вам когда-нибудь встретится `short` и вы будете знать что это такое.

```
1 public class Main {
2
3     public static void main(String[] args) {
4         short number = 10000;
5         print("Max for Short is: " + Short.MAX_VALUE);
6         print("Min for Short is: " + Short.MIN_VALUE);
7     }
8 }
Main > main()
Run: Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
Max for Short is: 32767
Min for Short is: -32768
Process finished with exit code 0
```

2. Преобразование одного типа в другой

Итак, давайте представим такую ситуацию. У вас есть число типа short, но в нем хранится всего лишь 100. И вы можете подумать, а зачем мне тогда short если мне достаточно byte и работать с ним быстрее и места занимает меньше? И вы будете совершенно правы. Можем ли мы как-то переместить сто из одного контейнера в другой? Грубо говоря – переложить пуговицу из большого ящика в маленькую коробочку? Давайте попробуем.

```
public class Main {
    public static void main(String[] args) {
        short number = 100;
        byte small = number;
    }
}
Main > main()
Messages: Build x
! Information: java: Errors occurred while compiling module 'LearnJava'
! Information: javac 1.8.0_282 was used to compile java sources
! Information: 30.03.21 12:54 - Compilation completed with 1 error and 0 warnings in 523 ms
! Error:(5, 23) java: incompatible types: possible lossy conversion from short to byte
```

Мы говорили что Java строготипизированный язык и нельзя просто так брать и менять все. В сообщении об ошибке видим следующее – возможны потери при конверсии из short в byte. Что же делать? Нажимаем на красную лампочку (или Alt+Enter если ее не видно) и кастим одно ко второму: cast to byte. И сразу выведем на экран что получилось.

Как и ожидалось, 100. А все потому, что вся информация вместились. Представьте что вы выливаете воду из ведра в маленький ковшик. Если в ведре мало воды и она может уместиться в ковшике, то ничего не выльется. Но что если в ведре много воды? Давайте посмотрим на это.

```
1 public class Main {
2
3     public static void main(String[] args) {
4         short number = 100;
5         byte small = (byte) number;
6         System.out.println(small);
7     }
8 }
Main
Run: Main x /usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java
100
Process finished with exit code 0
```

```
1 public class Main {
2
3     public static void main(String[] args) {
4         short number = 10000;
5         byte small = (byte) number;
6         System.out.println(small);
7     }
8 }
Main > main()
Run: Main x /usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java
16
Process finished with exit code 0
```

Из 10000 осталось лишь 16? Seriously? А почему не 127? Ведь у byte максимальное значение 127. А вы пробовали полное ведро вылить в стакан? Он не будет полон, так как многое выльется оттуда. Именно поэтому кастить одни данные к другим опасно. Делайте это лишь в случае уверенности, что не потеряете данных.

Ок, мы рассмотрели случай, когда из большого контейнера кладем данные в мелкий, но что тогда с обратной ситуацией? Что будет, если мы например из int положим данные в double? Можно предположить, что ничего страшного не должно случиться, ведь если мы положим пуговицу в огромный ящик, то она должна легко там уместиться, не? Проверим!

Во-первых посмотрите на написание числа 2 миллиарда. Раньше я писал без разделителей и было сложно понять, сколько там нулей. А теперь я разделил их _ и все стало проще. Секунду, разделять нули с помощью символа _? Да, оказывается так можно.

Во-вторых посмотрите как высветила наша среда разработки переменную number. Если навести курсор то можно узнать, что толку от нее нет. Ведь далее мы кладем это число в еще больший контейнер, т.е. могли бы сразу так сделать и не тратить сил и времени на int. Ок.

```
1 public class Main {
2
3     public static void main(String[] args) {
4         int number = 2_000_000_000;
5         double big = number;
6         System.out.println(big);
7     }
8 }
```

Main > main()

Run: Main x

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

2.0E9

Process finished with exit code 0

И давайте тогда посмотрим на линию 5. Мы не кастовали как в случае с byte. Почему же? А потому что double вмещает в себя больше данных чем int. И приводить одно к другому не нужно. И в конце когда выводим в консоль то видим что не было потерь. 2 и 9 нулей, все в порядке.

Так что постарайтесь запомнить эту вещь, она вам пригодится в работе. Если вы видите, что ваша переменная меньше чем та, которую нужно использовать и среда разработки предлагает вам кастовать данные, то попробуйте поменять тип вашей переменной. Лучше захватить в памяти больше места, чем потерять драгоценные данные. Это не значит, что лучше всегда использовать double. Нет. Просто ответственнее подходите к выбору типа данных. То же самое и про int и long. Зачастую айдишники например выходят за рамки 2 миллиардов и тогда уж точно придется использовать long. Так что когда вы пишете код и вам приходит число, обязательно задайте вопрос, а какое максимальное значение оно может принимать?

И да, вам не нужно запоминать максимальные значения каждого типа. Вам просто заглянуть в их классы и посмотреть на константу MAX_VALUE.

И напоследок рассмотрим такой случай. Помните метод который умножал числа? А давайте посмотрим как он справится с большими числами.

Давайте перемножим миллион и 2 миллиона. Естественно, что полученный результат не влезет в int. И что же тогда мы получим? Минус 1 миллиард и чет там еще. Чаво? А именно то же что и получилось, когда мы пытались записать 10000 в byte. Так что же нам тогда делать? Ну, если результат перемножения может выйти за рамки типа int, то логично будет преобразовать результат к double. Так? Ну да.

```
1 public class Main {
2
3     public static void main(String[] args) {
4         System.out.println(multiply(1_000_000, 2_000_000));
5     }
6
7     private static int multiply(int number1, int number2) {
8         return number1*number2;
9     }
10 }

Main > main()

Run: Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
-1454759936

Process finished with exit code 0
```

```
1 public class Main {
2
3     public static void main(String[] args) {
4         System.out.println(multiply(1_000_000, 2_000_000));
5     }
6
7     private static double multiply(int number1, int number2) {
8         return (double) number1 * number2;
9     }
10 }

Main > multiply()

Run: Main x
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
2.0E12

Process finished with exit code 0
```

Заметьте, что мы кастанули первый `int` к `double`. Почему? Потому что нужно представить один из множителей `double`, перед тем как произвести вычисления. Если мы кастанем произведение, то опять будет ошибка. Попробуйте сами. `(double) number1` относится лишь к первому аргументу. Если написать `(double) (number1*number2)`, то сначала мы перемножим числа и получим минус 1 миллиард, а потом уже просто поменяем тип. Так неправильно, можете сами проверить.

Так что вот вам еще одна штука для запоминания. Если вы понимаете, что при математических или иных действиях, ваши данные могут выйти за пределы, то просто перестрахуйтесь и возьмите тип, который может в себя вместить больше.