

# Темы 1.2 – 1.5

**Примитивные типы данных. Арифметические выражения.**

**Представление данных в памяти.**

**Операторы отношения.**

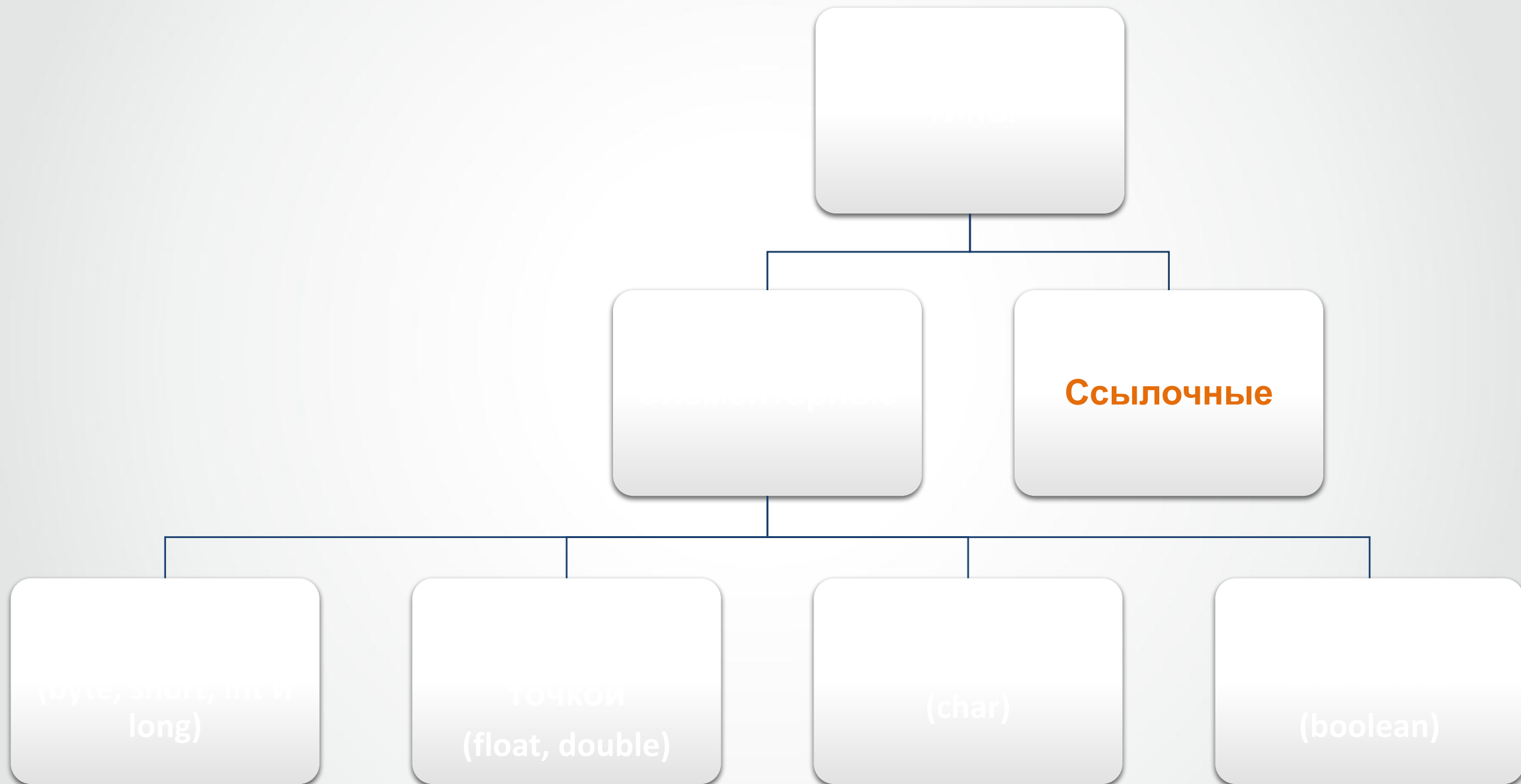
**Логические операторы.**

**Преобразования типов.**

**Побитовые (поразрядные) операторы**

## Java – строго типизированный язык

- Каждая переменная обладает типом, каждое выражение имеет тип и каждый тип строго определен.
- Все присваивания, как явные, так и за счет передачи параметров в вызовах методов, проверяются на соответствие типов.



**<тип> <имя переменной> [= <значение>];**

```
...  
int a2;  
double xx = 0.15;  
...
```

**Переменная может называться любой последовательностью латинских букв, цифр и знаков подчеркивания но, при этом, не могут начинаться с цифры.**

**Константа** — это именованная ячейка памяти, способная хранить данные, которые потом изменяться не будут.

**Константа** — это фактически переменная, объявленная с ключевым словом (модификатором) `final` (оно как раз и говорит о том, что значение переменной изменяться не будет).

```
...  
final double Pi = 3.1415926536;  
...  
Pi = 3.14; //Ошибка!  
...
```

## Все целые типы в Java знаковые!

Имя	Ширина в битах	Диапазон допустимых значений
<code>byte</code>	8	
<code>short</code>	16	
<code>int</code>	32	
<code>long</code>	64	

Проведем исследование на примере числа  $111000_2$  ( $56_{10}$ )

## Прямой код

0	0	1	1	1	0	0	0
7	6	5	4	3	2	1	0

**Положительный целые числа в прямом обратном и дополнительном коде выглядят одинаково!**

Проведем исследование на примере числа  $-111000_2$  ( $-56_{10}$ )

## Прямой код

<b>1</b>	0	1	1	1	0	0	0
7	6	5	4	3	2	1	0

## Обратный (инверсный) код

<b>1</b>	1	0	0	0	1	1	1
7	6	5	4	3	2	1	0

## Дополнительный код (двоичное дополнение)

<b>1</b>	1	0	0	1	0	0	0
7	6	5	4	3	2	1	0



Число	Код двоичного представления в byte (8 бит)		
	Прямой	Обратный	Дополнительный
127	01111111	01111111	01111111
1	00000001	00000001	00000001
0	00000000	00000000	00000000
<b>-0</b>	<b>10000000</b>	<b>11111111</b>	<b>не существует</b>
-1	10000001	11111110	11111111
-127	11111111	10000000	10000001
<b>-128</b>	<b>не существует</b>	<b>не существует</b>	<b>10000000</b>

Литерал (англ. *literal* — константа) — запись в ИСХОДНОМ коде компьютерной программы, представляющая собой фиксированное значение. Литералами также называют представление значения некоторого типа данных.

```
...  
int a = 012; //a=128=1010  
int b = 0x12; //a=1216=1810  
int c = 0b1101; //a=11012=1310  
...  
long L = 999999999999999999999999L;  
...  
long r = 0b1111_1111_1010_1111_1110_1111_0010_1110_1010_1101L;  
...
```

Операция	Пример
+	<pre>Int a = 9; a=a+4; //a=9+4=13</pre>
-	<pre>Int b = 9; b=b-4; //b=9-4=5</pre>
/	<pre>Int c = 9; c=c/4; //c=9/4=2</pre>
*	<pre>Int d = 9; d=d*4; //d=9*4=36</pre>
%	<pre>Int e = 9; e=e%4; //d=9%4=1</pre>

Операция	Пример
++	<p><b>Постфиксная запись</b></p> <pre>Int f = 9; Int g; g=f++; //g=9, f=10</pre> <p><b>Префиксная запись</b></p> <pre>Int f = 9; Int g; g=++f; //g=10, f=10</pre>
--	<p>-----     -----</p>
+=; -=; *=; /=; %=	<pre>Int a = 9; a+=4; //a=a+4=9+4=13</pre>

Написать программу, которая вычисляет количество миль, проходимым лучом света за указанное число дней. (приблизительная скорость света в милях за секунду - 186000)  
(в качестве типа для переменной результата использовать

```
final int v = 186000; //Приблизительная скорость света в милях за секунду
int d = in.nextInt(); //Ввод числа дней пользователем
long r; //Переменная для записи бедующего результата

long c = d*24*60*60; //Конвертируем количество дней в секунды
r = c*v; //Вычисляем количество миль(результат)

out.println(r); //Вывод результата
```

Имя	Ширина в битах	Диапазон допустимых значений
<code>float</code>	32	
<code>double</code>	64	

## В Java реализован стандарт IEEE-754

Имя	Ширина в битах	Диапазон допустимых значений
float	32	от $4.9e-324$ до $1.8e+308$
double	64	от $1.4e-045$ до $3.4e+038$

Имя	Ширина в битах	Знак	Мантисса	Экспонента (порядок)	Сдвиг
float	32	1	23	8	-127
double	64	1	52	11	-1023

```
float a = 1.234;  
float b = 6.022E23; //b=6.022*1023  
float c = 6.022E-23; //c=6.022*10-23  
float d = 6.02e+23; //d=6.022*10+23=6.022*1023  
float e = 0x12.2; //e =12.216= 18,125  
float f = 0x12.2p2; // f = 12.216 * 22 = 72.5  
...  
float g = 1.234f;  
double h = 1.234d;  
...  
float i = 9_999_999.1_0_9f;
```



Составить и отладить программу для вычисления площади круга.

Радиус вводится пользователем с клавиатуры.

(Формула площади круга -  $S = \pi r^2$  )

(Число Пи принять равным -  $\pi = 3.1416$ )

**(Помните!** В Java отсутствует оператор возведения в степень.)

```
final double Pi =3.1416; //Число ПИ
float r = in.nextFloat(); //Ввод пользователем радиуса круга
double s; //Переменная для сохранения площади
круга

s = Pi*r*r; //Вычисление площади круга (результата)

out.println(s); //Вывод результата
```

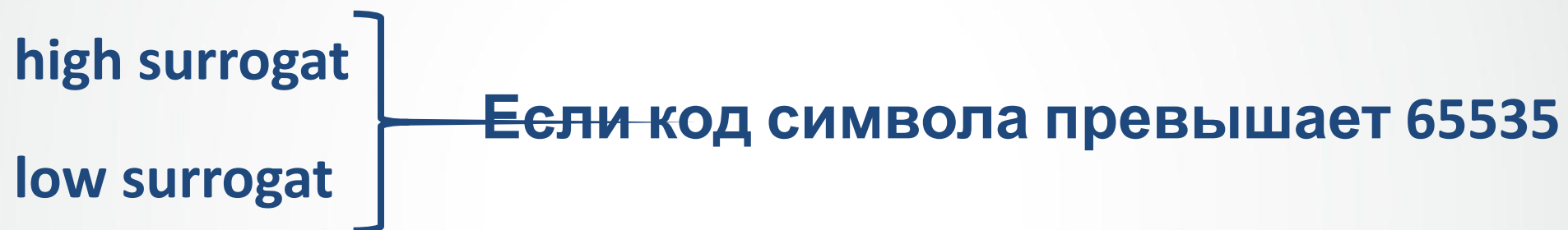
## Char не является знаковым типом!

Имя	Ширина в битах	Диапазон допустимых значений
char	16	

Char являет собой десятичное представление шестнадцатеричного кода символа в таблице Unicode.

Char свободно конвертируется в числовые типы и обратно.  
(доступны все арифметические и побитовые операции)

## Суррогатные пары



**Сегодня кодировка Unicode насчитывает около 110 000 СИМВОЛОВ**

```
char a = 'ë';
```

```
char b = '\u0804';
```

```
char b1 = 0x0804;
```

```
char c = '\141';
```

```
char c2 = 141;
```

```
char e = '\t'; // табуляция – код 9
```

```
char f = '\n'; // перевод каретки на следующую строку – код 10
```

```
char g = '\r'; // возврат каретки – код 13
```

```
char h = '\\"'; // одинарная кавычка
```

```
char i = '\\'; // обратный слеш
```

```
char j = '\\"'; // двойная кавычка
```

```
char k = '\b'; // возврат на одну позицию («забой»)
```

# Boolean НЕ преобразуется в любые другие простые

## ТИПЫ!

Операции сравнения:

<; >; ==; <=; >=; !=

Логические операции:

and	or	xor	not
&&		^	!
&			
&=	=	^=	

```
boolean a = true;
```

```
boolean b = false;
```

**переменная = выражение1 ? выражение2 :  
выражение 3**

- **выражение1 – boolean**
- **если выражение1 = true, то вычисляется выражение2**
- **если выражение1 = false, то вычисляется выражение3**
- **результат работы оператора равен значению вычисленного выражения**
- **выражение2 и выражение3 должны возвращать значения одинакового или совместного типа, которым не может быть void.**

Составить и отладить программу, которая, принимая от пользователя значения  $a, b$  и  $x$  определяет, лежит ли  $x$  внутри отрезка  $(a, b)$ . В случае попадания  $x$  в отрезок вывести `true`, в противном случае – `false`.

**(Для проверки условия использовать троичный условный оператор)**

```
int a = in.nextInt();           // ввод a
int b = in.nextInt();           // ввод b
int x = in.nextInt();           // ввод c
boolean r;                       // переменная для записи
результата
r = (x > a)&&(x < b)?true:false; // вычисление результата
out.println(r);                 // вывод результата
```



При присвоении данных переменной одного типа переменной другого типа выполняется автоматическое преобразование типа в случае удовлетворения следующих условий:

- оба типа совместимы
- длина текущего типа **больше** длины исходного типа

byte > short > int > long

float > double

...

```
byte b = 10;
```

```
short s = b;
```

...

## Преобразование char в int или long

```
...  
char a = 'a';  
int b = a;  
...
```

## Преобразование целочисленных типов в типы с плавающей точкой (возможна потеря точности)

```
...  
int a = 12;  
float b = a;  
...
```

## Оператор приведения типа (typename)

- При приведении более емкого целого типа к менее емкому старшие биты просто отбрасываются.
- При приведении дробных чисел к целым, дробная часть отбрасывается (без округления).

...

```
int i = (int) 3.94; //i=3
```

...

При вычислении выражения (a @ b), где @ - любая операция, a и b приводятся к одинаковому типу.

- если хоть одно число double, то к double
- если хоть одно число float, то к float
- если хоть одно число long, то к long
- в любом другом случае приводятся к int

```
...  
byte a = 3;  
byte b = a*3; //Ошибка!  
...
```

```
...  
byte a = 3;  
byte b=3*3; // Норма!  
...
```

Выражения (a @= выражение), где @ любая операция,  
раскрывается как  
a = (typename) (a @ выражение)

Неявно срабатывает приведение типов даже с потерей  
точности!

```
...  
int a = 300;  
byte b = 3;  
b += a //Норма!  
...
```

```
...  
int a = 300;  
byte b = 3;  
b = b + a //Ошибка!  
...
```

Могут применяться к целочисленным типам long, int, short, byte,

Операция	Описание
~	Побитовый унарный оператор NOT (НЕ)
&	Побитовое AND (И)
	Побитовое OR (ИЛИ)
^	Побитовое исключающее XOR
>>	Сдвиг вправо SHR (арифметический сдвиг)
>>>	Сдвиг вправо без учета знака SHR (логический сдвиг)
<<	Сдвиг влево SHL

Применяются ОТДЕЛЬНО К КАЖДОМУ БИТУ ЧИСЛА!

Является унарным (применяется к одной переменной)!

Инвертирует (меняет значения на противоположные) все биты операнда

<b>A</b>	<b>45</b>	<b>0000 0000 0000 0000 0000 0000 0010 1101</b>
<b>~A</b>	<b>-46</b>	<b>1111 1111 1111 1111 1111 1111 1101 0010</b>

...

```
int a = 45;
```

```
int b;
```

```
b = ~a; //b= -46
```

...

Составить и отладить программу, которая, позволяет пользователю получить противоположное значение введенного числа.

**(Для решения задачи используйте оператор побитового отрицания)**

```
int a = in.nextInt(); //Ввод числа пользователем
int r; //Переменная для записи результата
r = ~a+1; //Получение противоположного числа
out.println(r); //Вывод результата
```



Является бинарным (применяется к двум переменным)!

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

<b>A</b>	<b>45</b>	<b>0000 0000 0000 0000 0000 0000 0010 1101</b>
<b>B</b>	<b>17</b>	<b>0000 0000 0000 0000 0000 0000 0001 0001</b>
<b>A&amp;B</b>	<b>1</b>	<b>0000 0000 0000 0000 0000 0000 0000 0001</b>

...

```
int a = 45;
```

```
int b = 17;
```

```
int c;
```

```
c = a&b; //c = 1
```

...

Составить и отладить программу, которая, позволяет установить, является ли число степенью двойки. Если да, то вывести true, в противном случае false.

**(Для решения задачи используйте оператор побитовой конъюнкции)**

**(Для проверки условия использовать троичный условный**

```
int a = in.nextInt();           //ввод числа пользователем
boolean r;                       //переменная результата
r = (a != 0)&&((a & a - 1)==0)?true:false; //получение результата
out.println(r);                 //вывод результата
```

**На первый взгляд решение может показаться странным, но выполнив все действия с битами вручную, Вы наверняка поймете как это работает!**

Является бинарным (применяется к двум переменным)!

A	B	A B
0	0	0
0	1	1
1	0	1
1	1	1

<b>A</b>	<b>45</b>	<b>0000 0000 0000 0000 0000 0000 0010 1101</b>
<b>B</b>	<b>17</b>	<b>0000 0000 0000 0000 0000 0000 0001 0001</b>
<b>A B</b>	<b>61</b>	<b>0000 0000 0000 0000 0000 0000 0011 1101</b>

...

```
int a = 45;
```

```
int b = 17;
```

```
int c;
```

```
c = a|b; //c = 61
```

...

Является бинарным (применяется к двум переменным)!

A	B	$A \wedge B$
0	0	0
0	1	1
1	0	1
1	1	0

<b>A</b>	<b>45</b>	<b>0000 0000 0000 0000 0000 0000 0010 1101</b>
<b>B</b>	<b>17</b>	<b>0000 0000 0000 0000 0000 0000 0001 0001</b>
<b>A^B</b>	<b>61</b>	<b>0000 0000 0000 0000 0000 0000 0011 1100</b>

...

```
int a = 45;
```

```
int b = 17;
```

```
int c;
```

```
c = a^b; //c=60
```

...

Составить и отладить программу, которая, позволяет обменять значения двух переменных.

**(Для решения задачи используйте оператор XOR)**

**(Для решения задачи используйте только две переменные)**

```
int a = in.nextInt();  
int b = in.nextInt();  
a=a ^ b;  
b=a ^ b;  
a=a ^ b;  
out.println(a);  
out.println(b);
```

**Для лучшего понимания работы программы, выполните все операции вручную!**



# Сдвиг в право SHR (арифметический сдвиг)



переменная1 = переменная2 >> на сколько бит  
сдвинуть сдвиг с учетом знака

A	-46	1111 1111 1111 1111 1111 1111 1101 0010
A>>2	-12	1111 1111 1111 1111 1111 1111 1111 0100

...

```
int a = -46;
```

```
int b;
```

```
b=a>>2; //b = -12
```

...

**Составить и отладить программу, которая будет выполнять деление числа на 2.**

**(Использование операторов деления (/ и %) ЗАПРЕЩЕНО!)**

**(Для решения задачи используйте оператор побитового сдвига вправо с учетом знака)**

```
int a = in.nextInt();  
out.println(a >>= 1);
```

**Для лучшего понимания работы программы, выполните все операции вручную!**

# Сдвиг в право без учета знака SHR (логический сдвиг)



переменная1 = переменная2 >>> на сколько бит  
сдвинуть  
сдвиг без учета знака (левые позиции заполняются  
нулями)

A	-46	1111 1111 1111 1111 1111 1111 1101 0010
A>>2	1073741812	0011 1111 1111 1111 1111 1111 1111 0100

```
...  
int a = -46;  
int b;  
b = a>>>2; //b=1073741812  
...
```

переменная1 = переменная2 << на сколько бит  
сдвинута  
Правые позиции заполняются нулями

A	-46	1111 1111 1111 1111 1111 1111 1101 0011
A<<2	-180	1111 1111 1111 1111 1111 1111 0100 1100

...

```
int a = -46;
```

```
int b;
```

```
b = a<<2; //b=-184
```

...

**Составить и отладить программу, которая будет выполнять умножение числа на 2.**

**(Использование оператора умножения (\*) ЗАПРЕЩЕНО!)**

**(Для решения задачи используйте оператор побитового сдвига влево)**

```
int a = in.nextInt();  
out.println(a <<= 1);
```

**Для лучшего понимания работы программы, выполните все операции вручную!**

## Высший приоритет

(...)	[...]					
++ (префиксный)	-- (префиксный)	~	!	+	-	(приведение типов)
*	/	%				
+	-					
>>	>>>	<<				
>	>=	<	<=			
==	!=					
&						

Низший приоритет

↑Высший приоритет↑						
^						
&&						
?:						
=	op=					
++ (постфиксный)	-- (постфиксный)					
Низший приоритет						