

# Программирование на языке Java

## 8. Целые типы данных

# Программирование на языке Java

## Тема 8. Целые типы данных

# Типы данных

---

**Тип данных** – множество значений и набор операций, которые можно применять к этим значениям.

В математике множество целых чисел бесконечно, в компьютерных программах это не так.



Почему?

# Целые типы данных – 1

---

В Java 4 целых типа данных: `byte`, `short`, `int` и `long`.

Все целые типы в Java представляют значения со знаком – положительные и отрицательные.

**Ширина** целочисленного типа представляет собой занимаемый объем памяти.

# Целые типы данных – 2

---

Тип	Размер	Min	Max
<code>byte</code> (байт)	8 бит	-128	+127
<code>short</code> (короткое целое)	16 бит	$-2^{15}$	$2^{15}-1$
<code>int</code> (целое число)	32 бита	$-2^{31}$	$2^{31}-1$
<code>long</code> (целое число двойного размера)	64 бита	$-2^{63}$	$2^{63}-1$

# Объявление переменных целого типа

---

**Объявить переменную** – определить ее имя, тип, начальное значение, и выделить ей место в памяти.

```
public static void main()  
{  
byte a;  
short b, c;  
int d;  
long x=4, y, z;  
}
```

# Операции над величинами целого типа

---

+ – сложение

- – вычитание

\* – умножение

/ – деление нацело

% – получение остатка от деления

# Недопустимые операции

---

Деление на 0 и вычисление остатка от деления на 0 невозможны

```
int x = 1 / 0;
```

Ошибка времени выполнения

```
Exception in thread "main"  
java.lang.ArithmeticException: / by zero
```



# Особенность деления в Java

**!** При делении целых чисел остаток отбрасывается!

```
public static void main()  
{  
  int a = 7;  
  
  double x;  
  
  x = a / 4;  
  
  x = 4 / a;  
  
  x = (double)a / 4;  
  
  x = 1. * a / 4;  
  
  x = a / 4 * 1.;  
}
```

1.0

0.0

1.75

1.75

1.0

# Примеры выполнения операций / и %

---

$19 / 4$	$=$	$4$	$19 \% 4$	$=$	$3$
$-19 / 4$	$=$	$-4$	$-19 \% 4$	$=$	$-3$
$19 / (-4)$	$=$	$-4$	$19 \% (-4)$	$=$	$3$
$-19 / (-4)$	$=$	$4$	$-19 \% (-4)$	$=$	$-3$

# Определение цифр числа

**Задача.** Вывести на экран число сотен, десятков и единиц трехзначного числа.

```
public static void main()
{
    int x = in.nextInt();
    int one = x % 10;
    int dec = (x / 10) % 10;
    int hun = (x / 100) % 10;
    System.out.printf(".....");
}
```

456

6

45

4

5

4

# Целочисленные константы – 1

---

**Пример** целочисленных констант: 1, 2, 42, 93, ...

В числовых константах используются 4 вида представления:

- **десятичное**;
- **двоичное (начиная с Java 8)**  
обозначаются ведущим нулем и символом В:  
0b1001, 0B11, **0b120**
- **восьмеричное**  
обозначаются ведущим нулем:  
054, 0123, **091**
- **шестнадцатеричное**  
обозначаются ведущим нулем и символом X:  
0X54, 0x1Ab, 0X91, **0xQwerty**

## Целочисленные константы – 2

---

Целочисленные константы создают значение типа `int`.

Для создания константы типа `long` компилятору нужно явно указать тип, для этого к константе дописывают строчную или прописную букву `l`.

```
long x, y;  
x = 0x7fffffffffffffffffffffffL;  
y = 923789344394779L;
```

## Целочисленные константы – 3

---

Начиная с Java 7 в описании константы можно использовать символы «\_» (подчеркивание)

```
long creditCardNumber = 1234_5678_9012_3456L;  
long socialSecurityNumber = 999_99_9999L;  
long hexBytes = 0xFF_EC_DE_5E;  
long hexWords = 0xCAFE_BABE;  
long maxLong = 0x7fff_ffff_ffff_ffffL;  
byte nybbles = 0b0010_0101;  
long bytes =  
    0b11010010_01101001_10010100_10010010;
```

## Целочисленные константы – 4

---

Символ «\_» (подчеркивание) можно использовать в любом месте, кроме следующих

- в начале или конец числа
- перед суффиксом **L**
- В позициях, где ожидается строка цифр

```
int x1 = _52;  
int x2 = 5_2;  
int x3 = 52_;  
int x4 = 5_____2;  
int x5 = 0_x52;  
int x6 = 0x_52;  
int x7 = 0x5_2;  
int x8 = 0x52_;
```

# Инкремент и декремент

---

**Инкремент** – операция, увеличивающая переменную на единицу.

```
x = 5;  
x++;
```

6

**Декремент** – операция, уменьшающая переменную на единицу.

```
x = 5;  
x--;
```

4

Инкремент и декремент работают быстрее, чем обычное прибавление единицы, т.к. для их вычисления используются отдельные низкоуровневые команды, выполняемые на аппаратном уровне.



# Префиксная и постфиксная формы

Оператор инкремента можно записывать с обеих сторон:

- **префиксная форма** (прекремент) – оператор «++» или «--» записывается перед переменной

```
++x;
```

```
--x;
```

```
y = ++x;
```

```
=
```

```
x = x+1;
```

```
y = x;
```

1. выполняется операция,
2. вычисляется результат.

- **постфиксная форма** (посткремент) – оператор «++» или «--» записывается после переменной

```
x++;
```

```
x--;
```

```
y = x++;
```

```
=
```

```
y = x;
```

```
x = x+1;
```

1. вычисляется результат,
2. выполняется операция.

# Префиксная и постфиксная формы

х	Выражение	Итоговое у	Итоговое х
5	$y = x++;$	5	6
5	$y = ++x;$	6	6
5	$y = x--;$	5	4
5	$y = --x;$	4	4

# Инкремент и декремент. Задание

Вычислите значение переменной

```
x = 5; p = 5; q = 5;
```

```
w = 5; k = 5;
```

```
p *= x++;
```

```
q /= ++x;
```

```
w += --x;
```

```
k += x--;
```

<b>x</b>	5
<b>p</b>	25
<b>q</b>	0
<b>w</b>	11
<b>k</b>	11

# Переполнение

---

В Java возможна ситуация переполнения

Тип `int` занимает 32 бита, минимум  $-2^{31}$ , максимум  $2^{31}-1$

```
int x = 2147483647;  
x++;
```

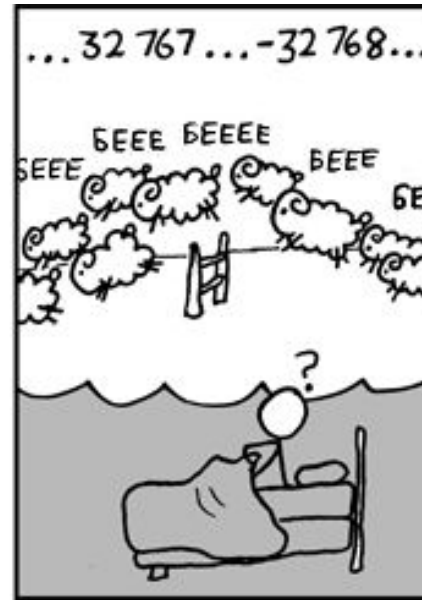
$2^{31} - 1$

$-2^{31}$

**Java не сообщит об ошибке переполнения!**

Будьте внимательны при работе с числами, близкими к максимальному или минимальному значению типа.

# Переполнение



# Задача

---

**Задача.** Отобразить текущее время в формате Часы:минуты:секунды, например 13:19:08, если метод `System.currentTimeMillis()` возвращает количество прошедших миллисекунд с начала эпохи Unix (01-01-1970 00:00:00) по Гринвичу

- 1 секунда = 1000 миллисекунд
- для нашего часового пояса нужно прибавить 9 часов
- Какой тип данных будем использовать?