



# Урок 2

## Типы данных

# Agenda

- Типы данных. Примитивные типы данных
- Объявление и инициализация переменных
- Литералы
- Строки
- Размещение данных в памяти: стек и куча
- Преобразование типов в Java
- Основные операторы в языке Java: арифметические, логические, битовые, строковые
- Базовая работа с консолью

# Типы данных

- Ссылочные типы данных
- Примитивные типы данных

# Типы данных

- Ссылочные типы данных
- Примитивные типы данных
- `null` тип

# Примитивные типы данных

## 8 примитивных типов данных:

### Целые числа

- byte - 8 бит
- short - 16 бит
- int - 32 бита
- long - 64 бита

### Числа с плавающей точкой

- float - 32 бита
- double - 64 бита

### Символы

- char - 16 бит, unicode

### Логические значения

- boolean

# Примитивные типы данных

## 8 примитивных типов данных:

### Целые числа

- byte - 8 бит
- short - 16 бит
- int - 32 бита
- long - 64 бита

### Числа с плавающей точкой

- float - 32 бита
- double - 64 бита

### Символы

- char - 16 бит, unicode

### Логические значения

- boolean - зависит от VM

# Примитивные типы данных

Q:  
Зачем?

# Примитивные типы данных

Q:  
Зачем?

A:

- Эффективность
- Переносимость



# Объявление и инициализация переменных

**Тип Идентификатор [= Значение] [, Тип Идентификатор [= Значение] ... ];**

- Тип это примитив или класс
- Идентификатор это имя переменной (см. java naming conventions)
- Значение это литерал (константа) или результат работы метода

# Литералы

## Целочисленные литералы:

- десятичная запись (1, 2, 10, 100500)
- восьмеричная запись (01, 05, 076)
- шестнадцатеричная (0xcafebabe)

# Литералы

## Целочисленные литералы:

- десятичная запись (1,2, 10, 100500)
- восьмеричная запись (01,05,076)
- шестнадцатеричная (0xcafebabe)
  
- двоичная (0b0101)
- символы подчеркивания (123\_0000\_3432, 0b0101\_1001)

# Литералы

## Литералы с плавающей точкой:

- стандартная запись (3.14)
- экспоненциальная запись (314E-02)
- символы подчеркивания (1\_12.5\_6\_7)

# Литералы

## Литералы с плавающей точкой:

- стандартная запись (3.14)
- экспоненциальная запись (314E-02)
- символы подчеркивания (1\_12.5\_6\_7)
- NaN (Not a Number), POSITIVE\_INFINITY, NEGATIVE\_INFINITY

# Литералы

## Литералы с плавающей точкой:

- стандартная запись (3.14)
- экспоненциальная запись (314E-02)
- символы подчеркивания (1\_12.5\_6\_7)
- NaN (Not a Number), POSITIVE\_INFINITY, NEGATIVE\_INFINITY

NaN != NaN

# Литералы

## Логические литералы:

- true
- false

# Литералы

## Символьные литералы:

- обычная форма ('a', 'b', 'c')
- восьмеричная форма ('\141')
- шестнадцатеричная форма ('\u0061')
- управляющие последовательности (\n,\r,\t и т.д.)



# Строки

**java.lang.String** - класс, представляющий собой строку в Java

- Строка это **объект**, а не просто массив символов
- Строка содержит в себе Юникод символы
- Строки **неизменяемы (имутабельны)**
- Для строковых литералов используют "" ("abc")

# Размещение данных в памяти: стек и куча

**Стек** это область памяти, связанная с Java потоком, хранящая в себе данные о вызове метода, его аргументы, локальные переменные и возвращаемое значение

# Размещение данных в памяти: стек и куча

**Стек** это область памяти, связанная с Java потоком, хранящая в себе данные о вызове метода, его аргументы, локальные переменные и возвращаемое значение

- Локальные примитивы и ссылки на объекты размещаются на стеке

# Размещение данных в памяти: стек и куча

**Стек** это область памяти, связанная с Java потоком, хранящая в себе данные о вызове метода, его аргументы, локальные переменные и возвращаемое значение

- Локальные примитивы и ссылки на объекты размещаются на стеке
- `StackOverflowError` при переполнении стека

# Размещение данных в памяти: стек и куча

**Стек** это область памяти, связанная с Java потоком, хранящая в себе данные о вызове метода, его аргументы, локальные переменные и возвращаемое значение

- Локальные примитивы и ссылки на объекты размещаются на стеке
- `StackOverflowError` при переполнении стека
- `OutOfMemoryError` при невозможности создать новый стек

# Размещение данных в памяти: стек и куча

**Куча (Heap)** это область памяти, связанная со всеми Java потоками одной JVM, хранящая в себе данные всех экземпляров классов и массивов

# Размещение данных в памяти: стек и куча

**Куча (Heap)** это область памяти, связанная со всеми Java потоками одной JVM, хранящая в себе данные всех экземпляров классов и массивов

- Создается JVM при старте

# Размещение данных в памяти: стек и куча

**Куча (Heap)** это область памяти, связанная со всеми Java потоками одной JVM, хранящая в себе данные всех экземпляров классов и массивов

- Создается JVM при старте
- Управляется автоматически с помощью сборщика мусора



# Размещение данных в памяти: стек и куча

**Куча (Heap)** это область памяти, связанная со всеми Java потоками одной JVM, хранящая в себе данные всех экземпляров классов и массивов

- Создается JVM при старте
- Управляется автоматически с помощью сборщика мусора
- `OutOfMemoryError` при переполнении кучи

# Преобразование типов в Java

- Автоматическое (расширяющее) преобразование
  - совместимость типов
  - длины целевого типа больше длины исходного
- Сужающее преобразование
  - (целевой\_тип) значение

# Основные операторы в языке Java

## Арифметические

- +, -, /, \*, %
- ++, -- (префиксная и постфиксная формы)
- +=, -=, /=, \*=, %=

# Основные операторы в языке Java

## Битовые

- $\sim$ ,  $\&$ ,  $|$ ,  $\wedge$
- $\gg$ ,  $\ggg$
- $\ll$
- $\&=$ ,  $|=$ ,  $\wedge=$
- $\gg=$ ,  $\ggg=$
- $\ll=$

# Основные операторы в языке Java

## Отношения

- ==, !=
- >, <
- >=, <=

# Основные операторы в языке Java

## Логические

- `!, &, |, ^`
- `&&, ||`
- `&=, |=, ^=`
- `==, !=`
- `?:`

# Основные операторы в языке Java

## Присваивание

- =

# Базовая работа с консолью

- **System.out** - экземпляр `PrintStream` класса,  
выводит данные на консоль
- **System.in** - экземпляр `InputStream` класса,  
читает данные с консоли



# Базовая работа с консолью

- **System.out** - экземпляр `PrintStream` класса,  
выводит данные на консоль
- **System.in** - экземпляр `InputStream` класса,  
читает данные с консоли
  - Оба потока уникальны
  - JVM создает их и ассоциирует с консолью
  - После закрытия потоков, их нельзя переоткрыть

# Home Work

- реализовать консольный калькулятор
  - i. операции  $+$ ,  $-$ ,  $*$ ,  $/$  и битовые
  - ii. форматированный вывод на консоль