HIGHER SCHOOL OF ECONOMICS
NATIONAL RESEARCH UNIVERSITY

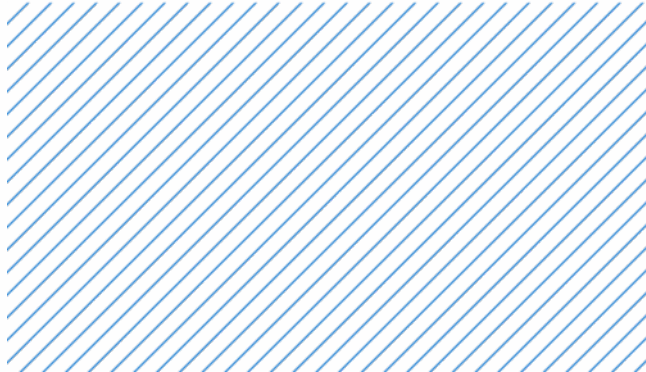
Лекция 3

Основы процедурного программирования на Java (часть 1)

Программирование на языке Java

Роман Гуров

ВШЭ БИ 2021



Простейшая программа на Java

Рассмотрим пример простейшей программы на языке Java – программу “Hello, world!”

Весь код должен храниться в файле, пусть у нас он называется `Main.java`

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Простейшая программа на Java

Рассмотрим пример простейшей программы на языке Java – программу “Hello, world!”

Весь код должен храниться в файле, пусть у нас он называется `Main.java`

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Объявляем класс Main – некоторую коробочку, в которой будут жить функции программы

В реальности, конечно же, смысл сильно отличается, но пока считайте так. Подробности в следующих сериях :)

Простейшая программа на Java

Рассмотрим пример простейшей программы на языке Java – программу “Hello, world!”

Весь код должен храниться в файле, пусть у нас он называется `Main.java`

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Ещё пара волшебных слов, необходимых для определения функции.

Простейшая программа на Java

Рассмотрим пример простейшей программы на языке Java – программу “Hello, world!”

Весь код должен храниться в файле, пусть у нас он называется `Main.java`

```
public class Main {
    public static void main(String[] args) {
        System.out.println( "Hello, world!" );
    }
}
```

Аргументы *программы*, которые можно передать при запуске, о них тоже пока сильно не думаем

Простейшая программа на Java

Рассмотрим пример простейшей программы на языке Java – программу “Hello, world!”

Весь код должен храниться в файле, пусть у нас он называется `Main.java`

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

А вот это нам уже вполне знакомо – это функция `main`, точка входа в программу

Всё остальное – воспринимайте как заклинание, попытки понять тщетны

Данная программа делает лишь одну вещь – при запуске печатает на экран фразу `Hello, world!`

`System.out.println` – встроенная функция языка Java для вывода на экран

Алфавит языка программирования – любые символы Юникода (хоть русские, хоть арабская вязь)
Но называйте переменные и функции по-английски.

Вывод в Java

```
System.out.println("Hello, world!");
System.out.print("Hello, world!");
```

- Версия с `ln` переносит строку в конце вывода, следующий принт будет с новой строки
- Версия без `ln` – не переносит, следующий принт будет в той же строке

В них можно передавать не только текстовые строки, но и разные другие типы данных:

```
System.out.println(14);
System.out.print("Важное число: ");
System.out.print(42);
System.out.println();
System.out.print(0.337);
System.out.println(" -- не очень важное число");
```

```
14
Важное число: 42
0.337 -- не очень важное число
```

Регистр важен

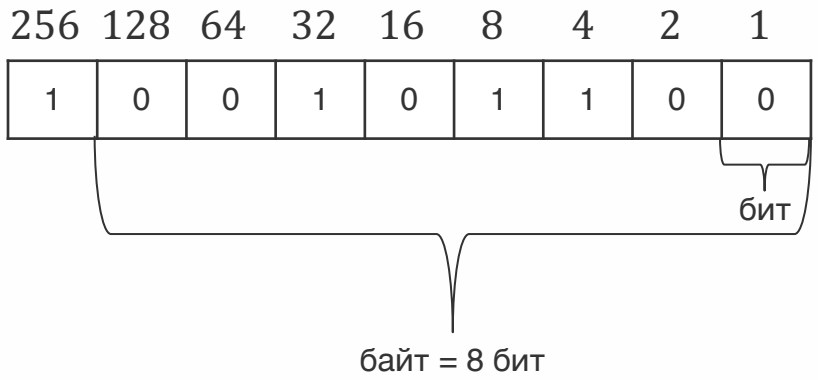
```
System.out.PrInTlN(1);
System.out.println(1);
```

Если в названии (напр. функции, переменной) поменять букву со строчной на заглавную (и наоборот), то получится абсолютно другое название

Двоичная арифметика

Компьютеры хранят все данные в двоичном виде: 0 и 1

Например, число 300_{10} в двоичной записи будет 100101100_2



Переменные в Java

В Java типы переменных делятся на две группы: примитивные и объектные

Примитивных типов всего восемь:

`int`, `byte`, `short`, `long`	Целочисленные типы
`float`, `double`	Вещественные типы
`boolean`	Булевский тип
`char`	Символьный тип

Каждое название примитивного типа – зарезервированное слово.

Примитивные типы

Переменная примитивного типа создаётся легко:

```java
public static void main(String[] args) {
    int a = 5;
    int b = a;
    a = 10;
    System.out.println(a);
    System.out.println(b);
}
```

```
10
5
```

Каждая переменная примитивного типа имеет свою конкретную ячейку в памяти
Две такие переменные не могут иметь одну общую ячейку

Численные типы

У каждого численного типа свой фиксированный размер в байтах
От размера зависит диапазон значений, которые может принимать переменная

Тип	Размер (в битах)	Размер (в байтах)	Мин	Макс
`byte`	8	1	`-128`	`127`
`short`	16	2	`-32768`	`32767`
`int`	32	4	`-2147483648`	`2147483647`
`long`	64	8	`-9223372036854775808`	`9223372036854775807`
`float`	32	4	$-3.4 \cdot 10^{38}$	$3.4 \cdot 10^{38}$
`double`	64	8	$-1.7 \cdot 10^{308}$	$1.7 \cdot 10^{308}$

Численные типы

Арифметика над численными типами вам уже знакома

```
public static void main(String[] args) {
    int y = 5;
    int x = y*y;
    x = x - y;
    x = x + 2;
    int answer = (10*x*x + 7*x - 3) / 42;
    System.out.println(answer);
}
```

```
118
```

Но стоит понимать, что арифметический оператор работает только с операндами одинакового типа

Более того, деление для целочисленных типов ведёт себя особенным образом:
Результат делается целым посредством отбрасывания дробной части

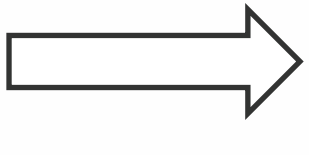
```
System.out.println(6/3);
System.out.println(7/3);
System.out.println(8/3);
System.out.println(9/3);
System.out.println(-5/2);
```

```
2
2
2
3
-2
```

Численные типы

Также, для чисел существует оператор %: взятие остатка от деления

```
System.out.println(3 % 3);
System.out.println(4 % 3);
System.out.println(5 % 3);
System.out.println(6 % 3);
```



```
0
1
2
0
```

Например, с его помощью удобно проверять числа на чётность: `x % 2`
Для четных вернётся `0`, для нечётных `1`

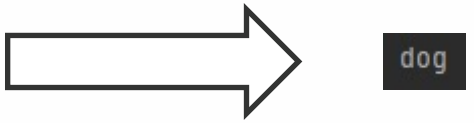
Символьный тип

Тип char хранит ровно один символ в кодировке UTF-16

Символ хранится как 16-битное целое число от 0 до $2^{16} - 1$

Для него тоже доступны арифметические операции

```
char c1 = 'd';
char c2 = 111;
char c3 = 'f' + 1;
System.out.print(c1);
System.out.print(c2);
System.out.println(c3);
```



```
dog
```

Символьный литерал записывается в одинарных кавычках
Это отличается от текстовых строк, для которых кавычки двойные

Булевский тип

Тип boolean хранит ровно одно из двух значений: `true`/`false` (истина/ложь)

Для хранения достаточно лишь одного бита:

`0 = false`
`1 = true`

```
boolean b1 = true;
boolean b2 = false;
System.out.println(b1);
System.out.println(b2);
```

```
true
false
```

Булевский тип и его операторы

В языке есть операторы сравнения, которые возвращают тип boolean

Оператор	Название	Пример
`==`	Равно	`x == y`
`!=`	Не равно	`x != y`
`>`	Больше	`x > y`
`<`	Меньше	`x < y`
`>=`	Больше или равно	`x >= y`
`<=`	Меньше или равно	`x <= y`

Для булевского типа существуют специальные *логические* операторы

Оператор	Название	Описание	Пример
`&&`	Логическое И	Возвращает `true`, если оба операнда `true`	`x < 5 && x < 10`
`||`	Логическое ИЛИ	Возвращает `true`, если хотя бы один операнд `true`	`x < 5 || x < 4`
`!`	Логическое НЕ	Обращает операнд, `true->false`, `false->true`	`!(x < 5 && x < 10)`

Примитивные типы

Итого, все примитивные типы:

Тип	Размер (в битах)	Размер (в байтах)	Мин	Макс
`byte`	8	1	`-128`	`127`
`short`	16	2	`-32768`	`32767`
`int`	32	4	`-2147483648`	`2147483647`
`long`	64	8	`-9223372036854775808`	`9223372036854775807`
`float`	32	4	$-3.4 \cdot 10^{38}$	$3.4 \cdot 10^{38}$
`double`	64	8	$-1.7 \cdot 10^{308}$	$1.7 \cdot 10^{308}$
`char`	16	2	0	65535
`boolean`	1	–	–	–

Приведение типов

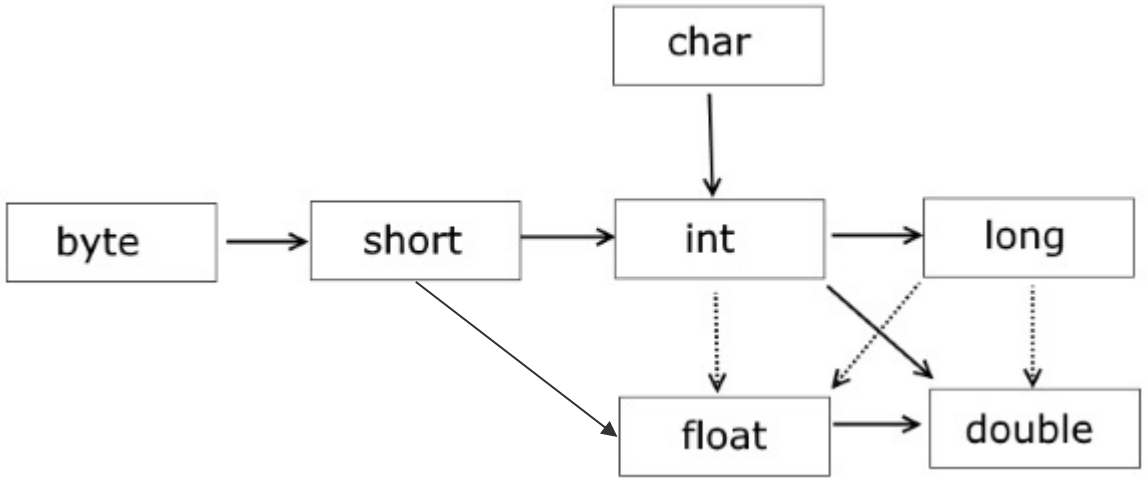
А как вообще подружить между собой разные типы?

Что произойдет, если умножить int на float?

Для таких ситуации применяется *преобразование типов*

```
byte b = 7;
int d = b;
```

В ситуациях, когда преобразование безопасно, оно происходит автоматически:



Приведение типов

Так что же произойдет, если умножить int на float?

Для преобразований операндов действует свод правил

- если один из операндов имеет тип `double`, то и второй операнд преобразуется к `double`
- иначе, если один из операндов имеет тип `float`, то и второй операнд преобразуется к `float`
- иначе, если один из операндов имеет тип `long`, то и второй операнд преобразуется к `long`
- иначе, все операнды преобразуются к типу `int`

```
int a = 3;
double b = 4.6;
double c = a + b;
```

А вот так уже не получится:

```
byte a = 3;
byte b = 5;
byte c = a + b;
```

Явное приведение типов

В остальных ситуациях, преобразование типа нужно указывать явно:

```
byte a = 3;
byte b = 5;
byte c = (byte)(a + b);
```

```
double a = 2.5;
int b = (int)a;
```

```
int cc = 66;
char c = (char)cc;
```

Объектные типы

Все типы, кроме примитивных, являются *объектными* (*ссылочными*)

Ввод в Java

Для ввода с клавиатуры нам понадобится особая сущность – Scanner

```
Scanner scanner = new Scanner(System.in);
int n = scanner.nextInt();
float f = scanner.nextFloat();
System.out.println(n * f);
```

Очередное заклинание…

Именно такое поведение мы представляли себе,
когда использовали функции вида `считать_число()`

Объектные типы

Все типы, кроме примитивных, являются *объектными* (*ссылочными*)

Scanner – объектный тип

```
Scanner scanner1 = new Scanner(System.in);
Scanner scanner2 = scanner1;
```

Обе переменных указывают на один и тот же объект

Создание объекта происходит с помощью слова new, а сама переменная хранит только ссылку на этот объект

Сам объект не принадлежит переменной, не лежит внутри переменной, объект где-то снаружи. Поэтому, две переменные могут ссылаться на один объект

if

Вспомним задачу о нахождении корня линейного уравнения

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    float a = scanner.nextFloat();
    float b = scanner.nextFloat();

    if (a != 0) {
        System.out.println(-a / b);
    } else if (b == 0) {
        System.out.println("x ∈ R");
    } else {
        System.out.println("Решений нет");
    }
}
```

В условии обязательно должен быть тип boolean (и у циклов тоже)

while с предусловием

Выведем числа от 1 до n

```
Scanner scanner = new Scanner(System.in);
int n = scanner.nextInt();

int i = 1;
while (i <= n) {
    System.out.println(i);
    ++i;
}
```

Оператор ++ называется инкрементом, является короткой записью `i += 1`

Аналогично, есть и декремент – оператор `--`

while с постусловием

Читаем числа с клавиатуры, пока не встретим -1

```
int n;

do {
    n = scanner.nextInt();
} while (n != -1);
```

for

Вновь выведем числа от 1 до n, но уже циклом for

```
Scanner scanner = new Scanner(System.in);
int n = scanner.nextInt();

for (int i = 1; i <= n; ++i) {
    System.out.println(i);
}
```

Интересно, что в цикле for можно не указывать ни одного параметра:

```
int i = 1;
for (;;) {
    if (i > n) {
        break;
    }
    System.out.println(i);
    ++i;
}
```

При этом, каждый из параметров можно реализовать вручную

`break` – специальное слово, вызывающее преждевременный выход из цикла

for

Вновь выведем числа от 1 до n, но пропустив число 5

```
for (int i = 1; i <= n; ++i) {
    if (i != 5) {
        System.out.println(i);
    }
}
```

А можно сделать чуть-чуть наоборот:

```
for (int i = 1; i <= n; ++i) {
    if (i == 5) {
        continue;
    }
    System.out.println(i);
}
```

`continue` – ещё одно специальное слово, преждевременно заканчивающее *итерацию* цикла (в while тоже)