

Повторение курса Basic



ПРЕПОДАВАТЕЛЬ



Юрий Костяной

Java/Kotlin backend-разработчик

- 3+ года опыта в коммерческой разработке
- 2+ года опыта в преподавании
- Проекты по интеграции сторонних платформ, CRM
- Проблемно-ориентированный подход в преподавании



ВАЖНО:

- Камера должна быть включена на протяжении всего занятия.
- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

ПЛАН ЗАНЯТИЯ

1. Содержание
2. Основной блок
3. Вопросы по основному блоку
4. Домашняя работа

1

ОСНОВНОЙ БЛОК

Mission impossible



Для чего мы учим Java?

Чтобы быть профессионалами



УЧИТЬ
ПРОГРАММИРОВАНИЕ
ДЛЯ БУДУЩЕЙ РАБОТЫ

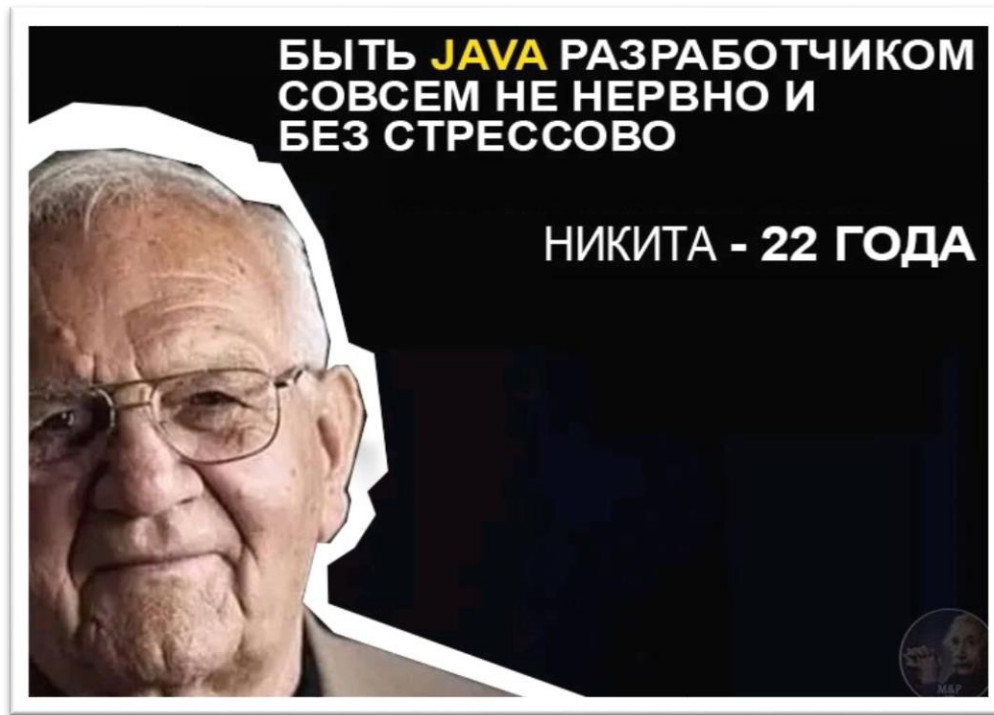


УЧИТЬ
ПРОГРАММИРОВАНИЕ
ЧТОБЫ ПОНИМАТЬ
ПРОГРАММИСТСКИЕ
ШУТКИ



Для чего мы учим Java?

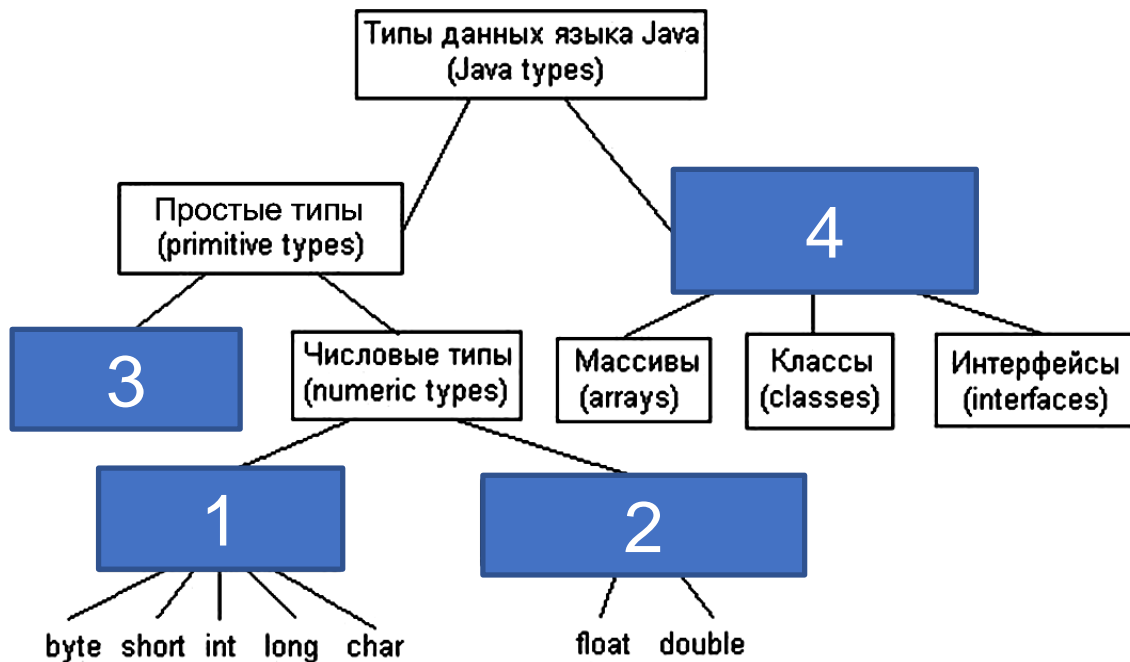
Чтобы не давать стареть мозгу



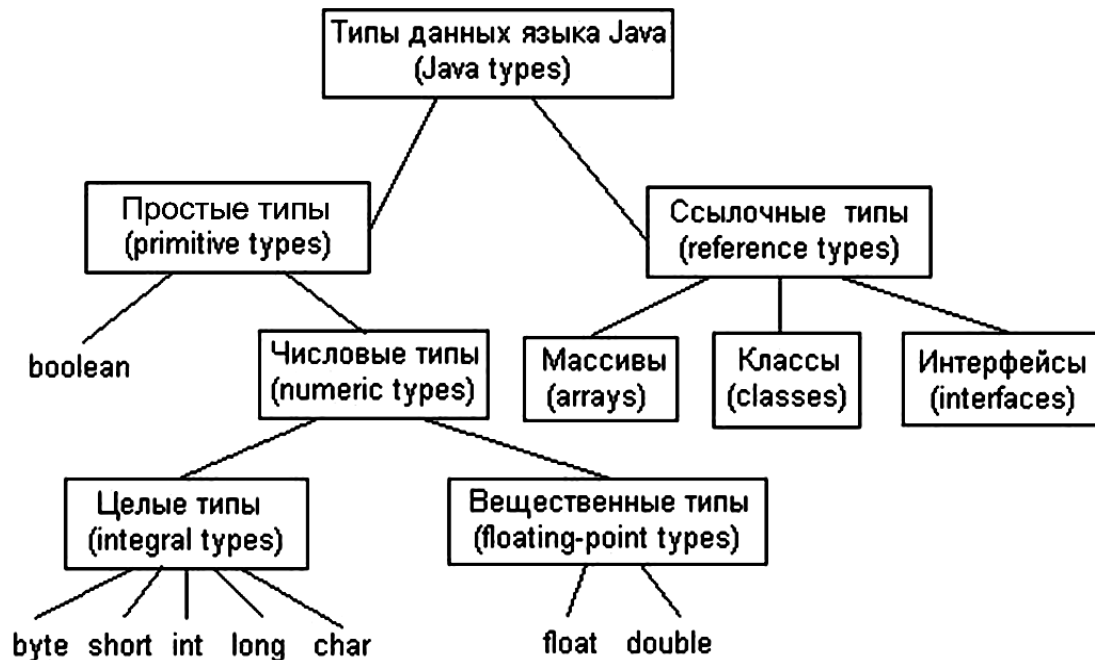
Подозрительный тип

Типы данных реального мира

Что скрыто за цифрами от 1 до 4?



Типы данных реального мира



Что скрыто за цифрами от 1 до 4?

<https://wordwall.net/resource/61010184>



С переменным успехом

Переменные

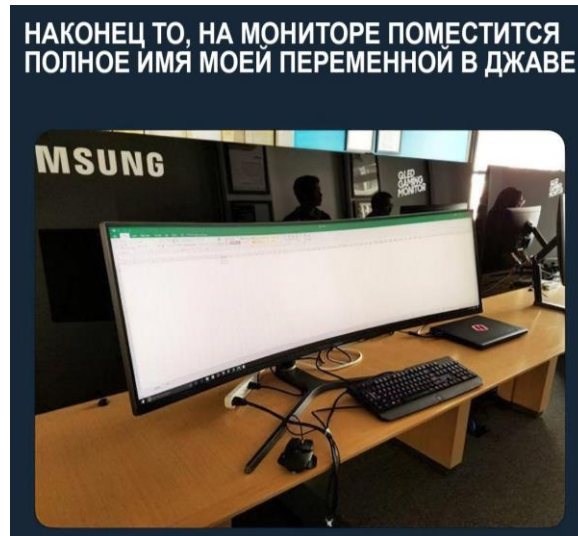
Переменная в Java состоит из типа и названия. Например, *int i*; *double d*; *String str*;

Опционально можно присвоить значение. Например, *int i2 = 10*; *Random rnd = new random()*;

Имена переменных должны:

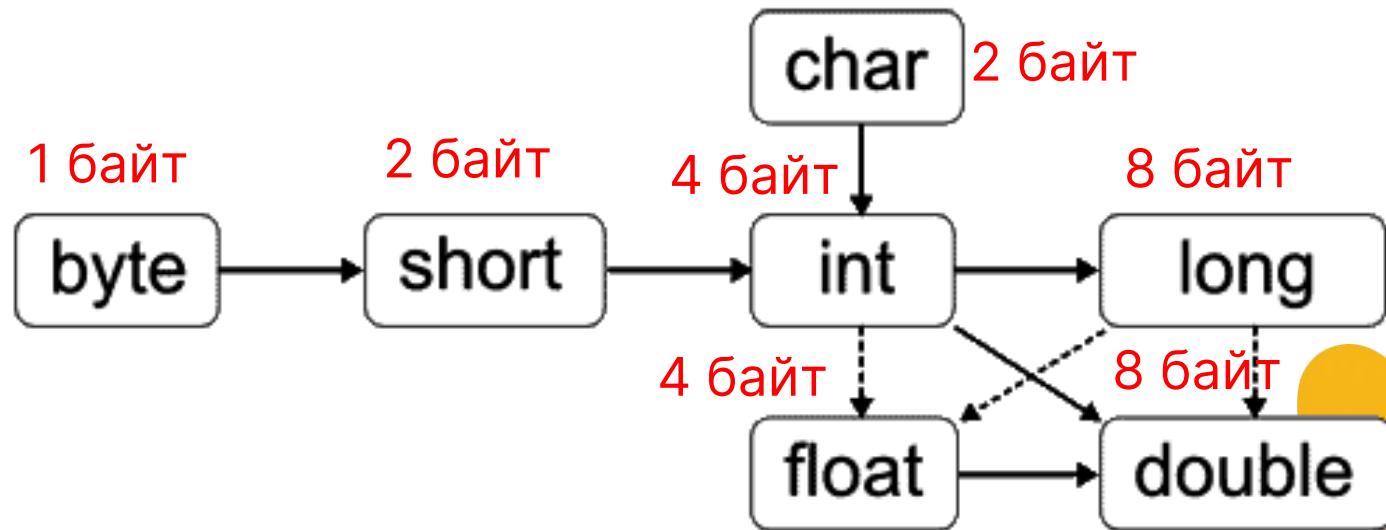
- начинаться со строчной буквы (могут начинаться также со знака подчёркивания «_», но так лучше не делать;
- могут содержать цифры, но цифра не может быть первой;
- состояться, используя стиль написания *camelCase*, т. е. каждое новое слово «склеивается» с предыдущим и записывается при этом с большой буквы;
- отображать смысл хранимых в ней данных (*age*, *name*, *weight* и т.д.), т.е. быть осмысленными;
- не содержать необоснованных сокращений;
- быть по возможности короткими;
- переменные типа *boolean* должны начинаться с префикса *is* или *has*.

В одном блоке кода, т.е. внутри пары фигурных скобок { }, имена переменных должны быть уникальны.



Каспер – доброе приведение

Приведение примитивных типов



A → B - A полностью помещается в B.

A→ B - A помещается в B с потерей точности.

В обоих случаях B помещается в A только частью своих значений (в пределах A) и требует **явного** приведения: *long lo = 1000l;*

int i = (int) lo;

Каспер – доброе приведение

Правила приведения

```
int i = 100;  
long lo = 1000L;  
double dou = 15.5;  
float f = 3.15f;  
char a = 'A';  
  
abc sum1 = i + lo;  
xyz diff = f - dou;  
qwe mul1 = i * f;  
xxx div = lo / f;  
yyy b = ++a;  
zzz c = a + 1;  
ttt d = (char) (a + 1);
```

Какой тип данных получится в результате операций над переменными (вместо *abc*, *xyz* и т.д.), если

- Приведение более узкого типа к более широкому происходит автоматически.
- Приведение более широкого типа к более узкому требует явного приведения (cast).
- При вычислении операции над значениями разных типов в результате получается больший тип.
- При вычислении операции над целочисленным значением и дробным в результате получается дробное.
- `char` числовым целочисленным типом. Каждому числу назначен символ.

Каспер – доброе приведение

Правила приведения

```
int i = 100;  
long lo = 1000L;  
double dou = 15.5;  
float f = 3.15f;  
char a = 'A';  
  
abc sum1 = i + lo; // long  
xyz diff = f - dou; // double  
qwe mul1 = i * f; // float  
xxx div = lo / f; // float  
yyy b = ++a; // char  
zzz c = a + 1; // int  
ttt d = (char) (a + 1); //char
```

Какой тип данных получится в результате операций над переменными (вместо *abc*, *xyz* и т.д.), если

- Приведение более узкого типа к более широкому происходит автоматически.
- Приведение более широкого типа к более узкому требует явного приведения (cast).
- При вычислении операции над значениями разных типов в результате получается больший тип.
- При вычислении операции над целочисленным значением и дробным в результате получается дробное.
- char числовым целочисленным типом. Каждому числу назначен символ.

Каспер – доброе приведение

А что же String?

Ссылочные типы тоже можно приводить, но по своим правилам.

Чтобы преобразовать любой примитив в строку используйте один из способов:

- добавление строки (это может быть даже пустая строка ""). Например, `int i = 1; String s = i + "";`
- метод `String.valueOf()`. Например, `int i = 1; String s = String.valueOf(i);`

Преобразовать строку в число или логический тип тоже возможно. Для этого используются методы классов-обёрток над примитивными типами:

```
String str = "65";  
byte b = Byte.parseByte(str);  
short sh = Short.parseShort(str);  
int i = Integer.parseInt(str);  
long l = Long.parseLong(str);  
char c = (char)Integer.parseInt(str);  
String boolStr = "true";  
boolean bool = Boolean.parseBoolean(boolStr);
```

Задание

<https://wordwall.net/resource/61010981>

Дана строка "10". Преобразуйте её в число и уменьшите на 1.

Из полученного числа и исходной строки составьте сообщение для вывода в консоль:

10 негрят отправилась обедать,

1 поперхнулся, их осталось 9.



Посчитаем

Простые математические операторы



Это математические операторы, которые можно использовать для выполнения различных простых или сложных арифметических операций над типами данных.

Сложение (+)

Этот оператор является бинарным и используется для добавления двух операндов.

Вычитание (-)

Этот оператор является бинарным и используется для вычитания двух операндов.

Умножение (*)

Этот оператор является бинарным и используется для умножения двух операндов.

Деление (/)

Это бинарный оператор, который используется для деления первого операнда (делимого) на второй операнд (делитель) и получения в результате частного.

Остаток от деления (%)

Это бинарный оператор, который используется для возврата остатка, когда первый операнд (делимое) делится на второй операнд (делитель).

Важно: в языке Java результат деления целого числа на целое – это всегда целое число, остаток при делении отбрасывается.

Посчитаем

Деление и остаток

В Java деление бывает трёх видов:

- обычное деление – в результате получаем вещественное число

```
5.0/2.0 // результат 2.5
```

- целочисленное деление – в результате даёт целое число. Остаток в этом случае отбрасывается.

```
5/2 // результат 2, т.к. остаток был отброшен
```

- целочисленное деление с получением остатка в виде целого числа – в результате даёт целое число из числителя рационального числа

```
5%2 // результат 1, т.к.  $5/2 = 2$  целых и  $1/2$ . Остаток - в числителе дроби
```

Задание

Вычислите результат деления

$$7 / 5 = ?$$

$$7.0 / 5 = ?$$

$$7 \% 5 = ?$$

$$123 / 100 = ?$$

$$123 \% 10 = ?$$

$$567 / 100 = ?$$

$$567 \% 10 = ?$$

Вопрос: как проверить, является ли число чётным/нечётным?

Вопрос: как проверить, является ли число кратным 3 или 5? (fizz-buzz)



Дайте данных!

Выбор случайных чисел

1 Импортировать класс *java.util.Random*

2 Создать экземпляр класса *Random*: с помощью оператора *new* создаём объект *Random* и присваиваем его переменной *rnd*.

Random rnd = new Random();

3 Вызовите один из следующих методов объекта *rnd*:

nextInt() генерирует случайные числа в диапазоне *int*.

nextInt(upper) генерирует случайные числа в диапазоне от 0 до *upper - 1*.

nextInt(low, up) генерирует случайные числа в диапазоне от *low* до *up - 1*.

nextFloat() генерирует число с плавающей запятой от 0.0 до 1.0.

nextDouble() генерирует двойное число между 0.0 и 1.0.



Дайте данных!

Ввод данных пользователем

За ввод данных пользователем отвечает класс *Scanner*.

1 Импортировать класс *java.util.Scanner*

2 Создать экземпляр класса *Scanner*:

```
Scanner scr = new Scanner(System.in)
```

System.in показывает, откуда объект *scr* будет брать данные. Этот класс умеет читать не только из консоли, но и читать из файла, например.

3 Вызовите один из следующих методов объекта *scr*:

nextInt() читает введённое целое число из консоли и распознаёт его как *int*.

nextDouble() читает введённое дробное число из консоли и распознаёт его как *double*.

nextLine() читает введённую строку.



Задание

Напишите программу «Средняя оценка».

Пользователь ставит оценку от 0 до 100. Программа ставит свою оценку случайным образом, а затем выводит, среднее значение из двух оценок.



Опера об операторах

Проблема

В каком порядке операторы будут вычислены в выражении?

```
int a = 1, b = 2, c = 3, d = 4;  
int e = a - b * c + b * d;
```



Опера об операторах

Проблема

В каком порядке операторы будут вычислены в выражении?

```
int a = 1, b = 2, c = 3, d = 4;  
int e = a - b * c + b * d;
```

- 1 рассчитает $b * c$
- 2 рассчитает $b * d$
- 3 вычислит разность (-)
- 4 вычислит сумму (+)



Опера об операторах

Проблема

В каком порядке операторы будут вычислены в выражении?

```
System.out.println("1" + 2 + 3);  
System.out.println(3 + 2 + "1");
```



Опера об операторах

Проблема

В каком порядке операторы будут вычислены в выражении?

```
System.out.println("1" + 2 + 3); // 123  
System.out.println(3 + 2 + "1"); // 51
```



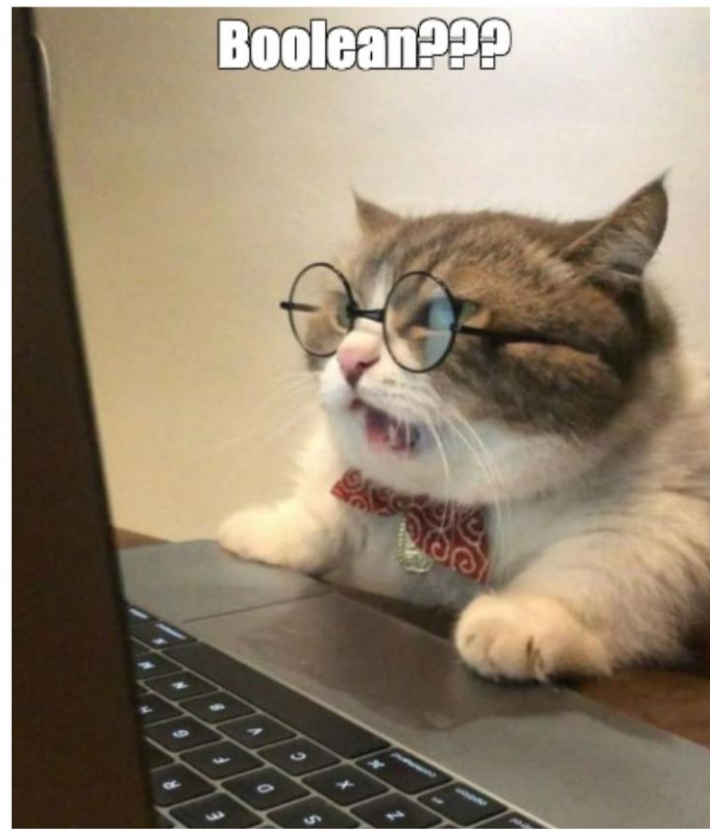
Да, нет, наверное

Как создать **boolean**

boolean – это примитивный тип данных, состоящий из двух значений – *true* или *false*.

Получить *boolean*-значение можно одним из следующих способов:

- 1 Создать явно: *boolean isEmpty = true;*
- 2 Сравнить два числа с помощью операторов сравнения `==`, `!=`, `<`, `>`, `<=`, `>=`
- 3 Сравнить два объекта с помощью `==` или метода *equals()*
- 4 Использовать логические операции над другими *boolean*: `!`, `&&`, `//`, `^`



Где логика?

Логические операторы

ТАБЛИЦА ИСТИННОСТИ

X1	X2	!X1	X1 X2	X1 && X2	X1 ^ X2
0	0	1	0	0	0
0	1	1	1	0	1
1	0	0	1	0	1
1	1	0	1	1	0

0 эквивалентен *false*, 1 эквивалентна *true*.

результат логических операций всегда будет 0 или 1.

! – НЕ (NOT), логическое отрицание, меняет значение на обратное: $!1 = 0$, $!0 = 1$.

|| – ИЛИ (OR), логическое сложение $1+1=1$, $0+0=0$, $0+1=1$.

&& – И (AND), логическое умножение $1*1=1$, $0*0=0$, $0*1=0$.

^ – исключающее ИЛИ (XOR), один или другой, но не оба:
 $1^1=0$, $0^0=0$, $1^0=1$

ДЕСЯТИЧНАЯ СИСТЕМА: $1+1=2$
ДВОИЧНАЯ СИСТЕМА: $1+1=10$
БУЛЕВАЯ АЛГЕБРА: $1+1=1$

НЕ ПРОГРАММИСТЫ:



Задание

Дополните программу «Средняя оценка».

Выведите фразу: *Have I passed the exam?*

В случае когда, средняя оценка больше 30 или пользователь ввёл оценку ниже 0, выведите *false*.

В случае когда, средняя оценка больше 50 и оценка компьютера при этом больше оценки пользователя, выведите *true*.



Ежели

Оператор логического перехода



```
//Программа что-то делает
if (условие1) {
    действия1 программы;
}
//Программа что-то делает
```

Условие внутри скобок if – это ни что иное, как значение *boolean*. Это может быть *boolean*-переменной или значение полученное любым другим способом.

```
//Программа что-то делает
if (условие1) {
    действия1 программы;
} else {
    действия2 программы;
}
```

Задание

Дополните программу «Средняя оценка».

Если пользователь набрал меньше 33 баллов, то вывести оценку 2.

Если пользователь набрал от 33 до 66 баллов, то вывести оценку 3.

Если пользователь набрал от 66 до 80 баллов, то вывести оценку 4.

Если пользователь набрал больше 80 баллов, то вывести оценку 5.



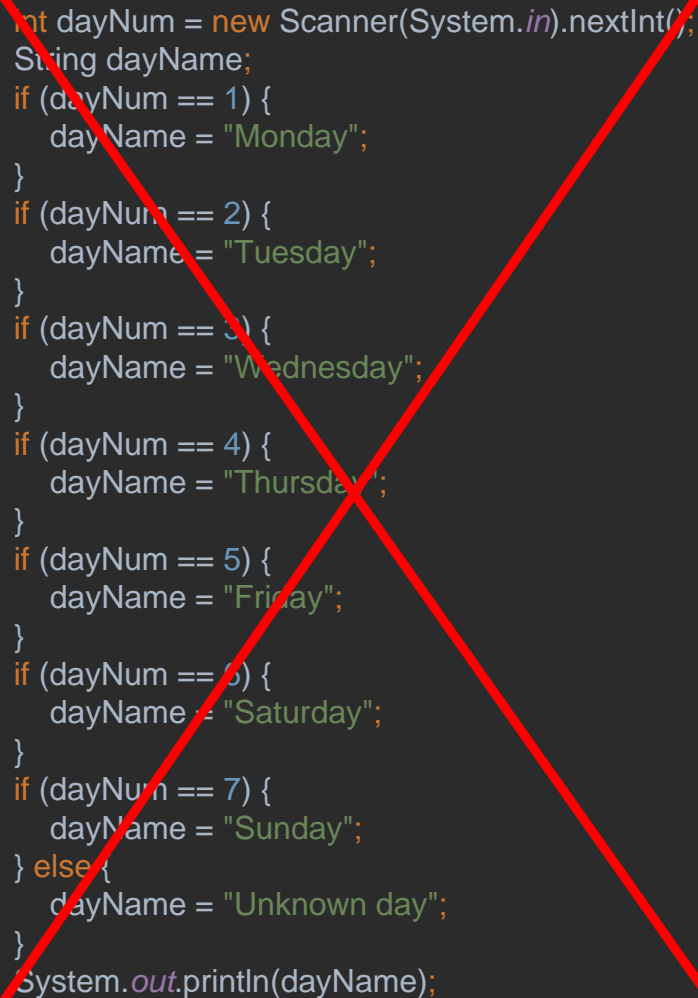
Выбиралка

Оператор switch

Когда нужно выбрать один вариант из нескольких лучше всего подходит конструкция *switch - case*.

```
int dayNum = new Scanner(System.in).nextInt();
String dayName = switch (dayNum) {
    case 1 -> "Monday";
    case 2 -> "Tuesday";
    case 3 -> "Wednesday";
    case 4 -> "Thursday";
    case 5 -> "Friday";
    case 6 -> "Saturday";
    case 7 -> "Sunday";
    default -> "Unknown day";
};
```

dayNum поочерёдно сравнивается со значениями, записанными после ключевого слова *case*. Если значения равны (*==*), то выбирается соответствующая ветка



```
int dayNum = new Scanner(System.in).nextInt();
String dayName;
if (dayNum == 1) {
    dayName = "Monday";
}
if (dayNum == 2) {
    dayName = "Tuesday";
}
if (dayNum == 3) {
    dayName = "Wednesday";
}
if (dayNum == 4) {
    dayName = "Thursday";
}
if (dayNum == 5) {
    dayName = "Friday";
}
if (dayNum == 6) {
    dayName = "Saturday";
}
if (dayNum == 7) {
    dayName = "Sunday";
} else {
    dayName = "Unknown day";
}
System.out.println(dayName);
```


Правила для операторов `switch`

- Может быть любое количество случаев *case*, но повторяющиеся значения случаев не допускаются.
- Значение для *case* должно иметь тот же тип данных, что и переменная в `()` у *switch*.
- Значение для *case* должно быть постоянным. Переменные не допускаются.
- Оператор **`break`** используется внутри старого *switch* для завершения последовательности операторов случая.
- Каждый оператор *case* может иметь оператор *break*, который является необязательным. Когда управление достигает оператора *break*, оно переходит к первой инструкции после всего *switch*. Если оператор *break* не найден, выполняется следующий случай.
- Оператор *default* не всегда является обязательным и может появляться в любом месте внутри блока переключателя. В случае, если он не в конце, то после оператора *default* необходимо оставить оператор *break*, чтобы пропустить выполнение следующего оператора *case*.

Задание

Дополните программу «Средняя оценка».

Назначьте оценке пользователя слово, описывающее оценку:

2 – неуд.

3 – удов.

4 - хорошо

5 – отлично.

Выведите слово в консоль.



Перечислялка

Перечисление

Перечисление (enum) – это ссылочный тип данных (разновидность класса), который имеет ограниченный набор значений (ограниченный набор объектов), причём доступ к каждому значению можно получить через имя класса.

DayOfWeek.java

```
public enum DayOfWeek {  
    MONDAY,  
    TUESDAY,  
    WEDNESDAY,  
    THURSDAY,  
    FRIDAY,  
    SATURDAY,  
    SUNDAY  
}
```

Использование перечисления DayOfWeek в классе Main (файл Main.java)

```
public class Main {  
    public static void main(String[] args) {  
        DayOfWeek today = DayOfWeek.THURSDAY;  
        DayOfWeek tomorrow = DayOfWeek.FRIDAY;  
        DayOfWeek weekend = DayOfWeek.SATURDAY;  
    }  
}
```

Задание

Дополните программу «Средняя оценка».

Создайте перечисление оценок по американской системе (A, B, C, D, E, F). Назначьте каждой оценке пятибалльной системы одно из значений американской системы оценок.

0 – F

1 – E

2 – D

3 – C

4 – B

5 – A.

Затем по полученной оценке американской системы, используя switch, выведите «pass», когда оценка равна A, B или C. Выведите «fail», когда оценка D. Выведите «no data», когда оценка E или F.

Когда стоит зацикливаться

Структура цикла

- 1 У любого цикла есть **тело** и **условие продолжения** выполнения.
- 2 Тело цикла заключено в { }. Это код, который будет повторяться.
- 3 Один прогон тела цикла называется **итерацией**.
- 4 Количество повторений ограничивается условием, которое проверяется до или после каждой итерации.



Когда стоит зацикливаться

Виды циклов Java

- **Цикл (loop)** - обеспечивает многократное выполнение набора инструкций, пока какое-то условие оценивается как истинное.
- Java предоставляет несколько способов организации циклов.

Цикл while

Работает, пока выполняется условие. `while(a>b) { }`

Цикл do while

Один раз выполняет тело цикла, потом проверяет условие. `do { } while(a>b);`

Цикл for

Имеет встроенный счётчик. Когда счётчик достигает определённого значения, цикл перестаёт работать. Шаг счётчика можно задавать. `for (int i = 0; i < 100; i++) { }`

Цикл foreach

Цикл по коллекции или массиву. Содержит переменную того же типа, что элемент массива. В переменную по очереди кладутся элементы из массива. В теле мы оперируем этой переменной.

Рекурсия

Вызов метода из тела того же метода приводит к зацикливанию программы.

Доколе

Синтаксис while

while читается как «пока <условие == true/истинно/выполняется> повторять код {...}»

```
while (условие продолжения) {  
    // код в теле цикла  
}
```

Условие – это значение типа boolean.

Пример:

```
int a = 0;  
while (a < 5) {  
    System.out.println("a = " + a);  
    a++;  
}  
System.out.println(«Конец программы»); // этот код уже вне тела цикла
```

Что будет, если вместо `int a = 0` будет `int a = 10`?

Доколе Бесконечный while

Чтобы сделать бесконечный цикл, нужно в условии указать истинное значение.

```
while (true) {  
    System.out.println("Фраза будет выводиться бесконечно");  
}  
System.out.println("Этот код не будет достигнут");
```



Куй железо, пока горячо

Синтаксис do-while

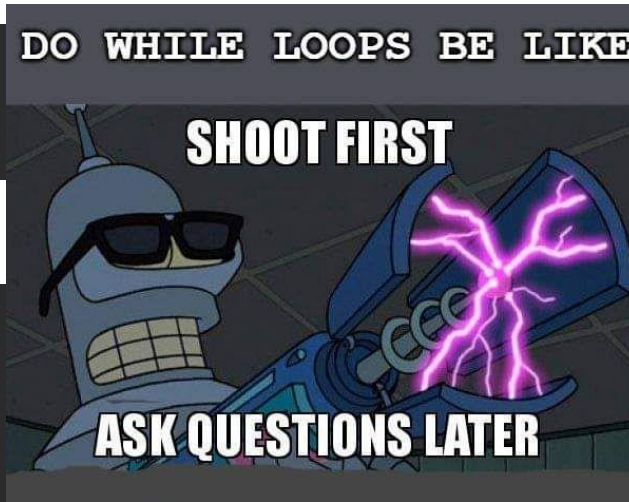
do-while читается как *повторять код {...}* «пока *<условие == true>*»

В отличие от *while* сначала выполнится код тела цикла, а затем будет проверка условия, т.е. тело выполнится минимум 1 раз

```
do {  
    // код в теле цикла  
} while (условие продолжения);
```

Пример:

```
int a = 0;  
do {  
    System.out.println("Повторим эту строку");  
    a++;  
} while (a < 5)  
System.out.println("Команда вне тела цикла выполняется только раз");
```



Что будет, если вместо *int a = 0* будет *int a = 10*; ?

Поставить на счётчик

Синтаксис цикла for?



`for` читается как «для <переменной> от 0 до N с шагом повторять код {...}, переменную изменять по правилу»

```
for (создание переменной счётчика; условие продолжения; как изменять переменную счётчика) {  
    // код тела цикла  
}
```

Пример:

```
for (int i = 0; i < 3; i++) {  
    System.out.println("Напишу эту фразу много раз!");  
}
```

Сколько раз надпись будет выведена?

Шорткат в IDE для `for` – это *fori*

Поставить на счётчик

Управление итерациями

Чтобы прервать выполнение любого цикла, используйте *break*:

```
int a = 0;
while (true) {
    System.out.println("Фраза будет выводиться 5 раз");
    if (a == 4 ) break;
    a++;
}
System.out.println("Этот код будет достигнут");
```

Если вместо *break* использовать *return*, то цикл прекратит работу, как и весь метод, в котором находится цикл. Если цикл находится в *main()*, то программа полностью завершит работу.

Чтобы пропустить итерацию, используйте *continue*:

```
int a = 0;
while (true) {
    if (a % 2 = 1 ) continue;
    System.out.println("Только чётные значения " + a);
    a++;
}
```



Для каждого

Цикл `foreach`

В Java есть краткая форма цикла *for*, которую, по аналогии с *C#*, принято называть *foreach*. Она используется для *коллекций и массивов*.

```
for (Тип переменной имяПеременной : коллекцияИлиМассив) {  
    // здесь идёт работа с переменной имяПеременной  
}  
/* Тип переменной должен совпадать с типом элементов коллекции/массива.  
Берём элемент коллекции, присваиваем его значение переменной имяПеременной.  
В цикле что-то делаем с имяПеременной. Цикл останавливается, когда больше нет элементов в  
коллекции */
```

Для каждого

Цикл `foreach`

```
public class Main {  
    public static void main(String[] args) {  
        for (java.time.Month month : java.time.Month.values()) {  
            System.out.print(month.ordinal() + " " + month.name() + " ");  
        }  
        /* java.time.Month - это Enum из пакета java.time стандартной библиотеки.  
        Полная запись обращения к классу java.time.Month позволяет не указывать import в файле */  
    }  
}
```

Main ×

```
"C:\Program Files\Java\jdk-17.0.5\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community  
0 JANUARY 1 FEBRUARY 2 MARCH 3 APRIL 4 MAY 5 JUNE 6 JULY 7 AUGUST 8 SEPTEMBER 9 OCTOBER 10 NOVEMBER 11 DECEMBER  
Process finished with exit code 0
```

Задание

Пользователь вводит порядковый номер месяца. Если введён неправильный номер, повторите запрос ввода номера месяца (do-while).

Используя enum Month из пакета java.time, выведите в консоль имена месяцев от January до месяца, введённого пользователем включительно (for).

Выведите в консоль все месяцы, кроме того, который указал пользователь (foreach).



Строки и строинги

Класс String

- **String** в Java – это объекты, внутри которых хранится массив байт и способ их интерпретации как символов (кодировка).
- Когда вносятся изменения в строку, создается совершенно новая строка.
- Когда объект *String* создается как литерал, объект будет создан в пуле констант String.
- С помощью оператора *new*. В случае динамического выделения строк им назначается новая ячейка памяти в куче. Эта строка не будет добавлена в пул констант String.

Варианты создания строк:

```
String str1 = "Java";
```

```
String str2 = new String(); // пустая строка
```

```
String str3 = new String(new char[] {'h', 'e', 'l', 'l', 'o'});
```

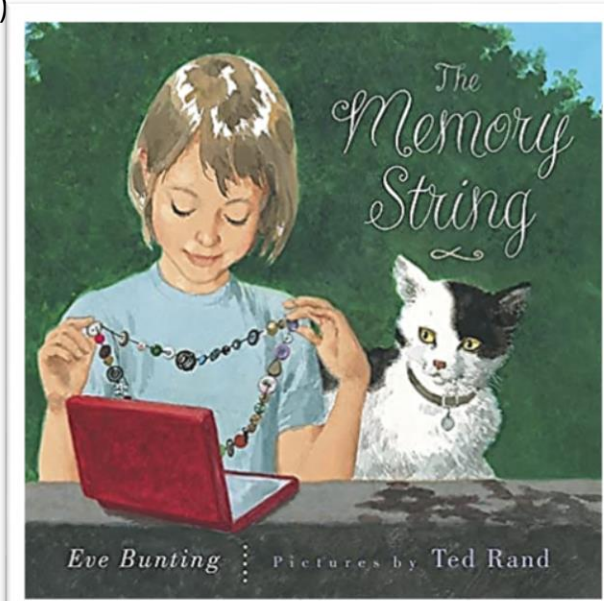
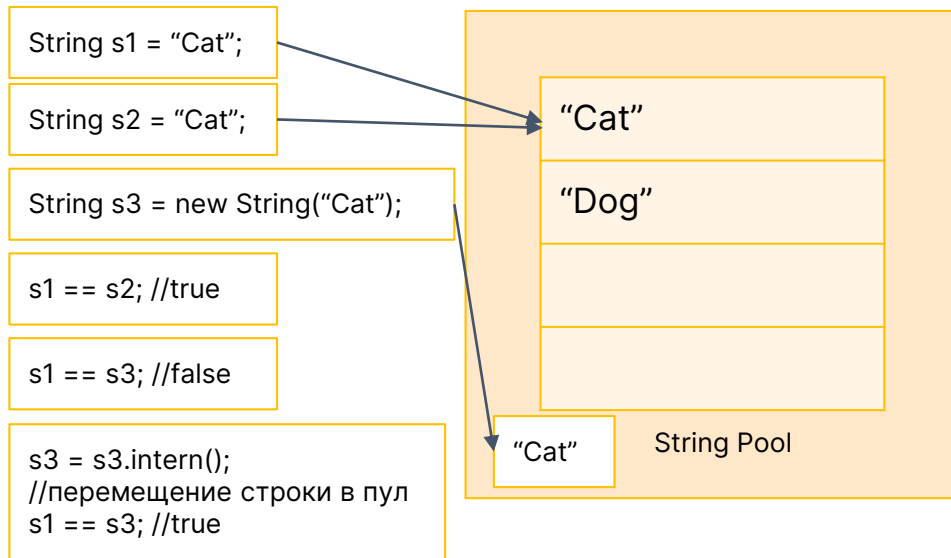
```
String str4 = new String(new char[] {'w', 'e', 'l', 'c', 'o', 'm', 'e'}, 3, 4);  
//3 -начальный индекс, 4 -кол-во символов
```



Строки и строинги

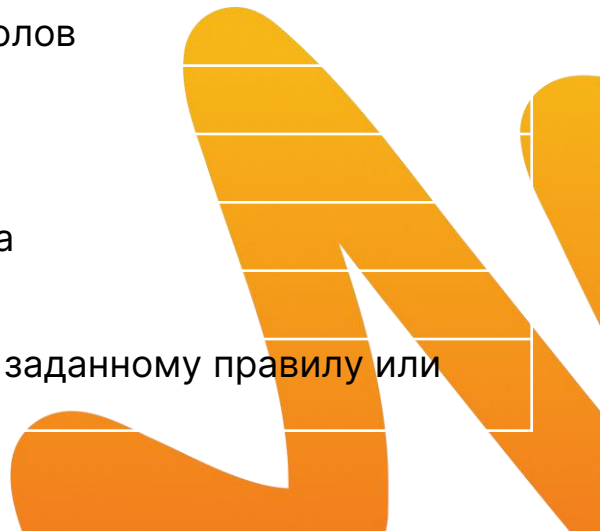
Строки в памяти

- Java использует концепцию строкового литерала.
 - более эффективно с точки зрения памяти
 - новые объекты не создаются, если они уже существуют в пуле строковых констант.
- `String s = new String("Hello!");`
 - новый строковый объект в обычной памяти "куче" (не в пуле).
- Пул строк (String Pool) - множество строк в куче (Java Heap Memory)



Что умеет String (методы)

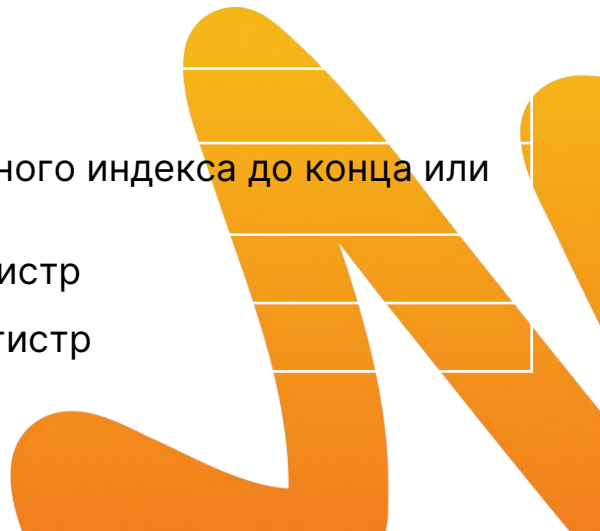
length()	получить длину строки (количество символов)
charAt()	получить символ в позиции index
isEmpty()	проверка «строка пуста?»
toCharArray()	преобразовать строку в массив символов
getChars()	возвращает группу символов
equals()	сравнивает строки с учетом регистра
equalsIgnoreCase()	сравнивает строки без учета регистра
regionMatches()	сравнивает подстроки в строках
split()	разделяет строку на массив строк по заданному правилу или разделителю



Строки и строинги

Что умеет String (методы)

indexOf()	находит индекс первого вхождения подстроки в строку
lastIndexOf()	находит индекс последнего вхождения подстроки в строку
startsWith()	определяет, начинается ли строка с подстроки
endsWith()	определяет, заканчивается ли строка на определенную подстроку
replace()	заменяет в строке одну подстроку на другую
trim()	удаляет начальные и конечные пробелы
substring()	возвращает подстроку, начиная с определенного индекса до конца или до определенного индекса
toLowerCase()	переводит все символы строки в нижний регистр
toUpperCase()	переводит все символы строки в верхний регистр



Строки и строинги

Что умеет String (методы JDK11+)



isBlank()

возвращает true, если строка пуста или содержит только пробелы

lines()

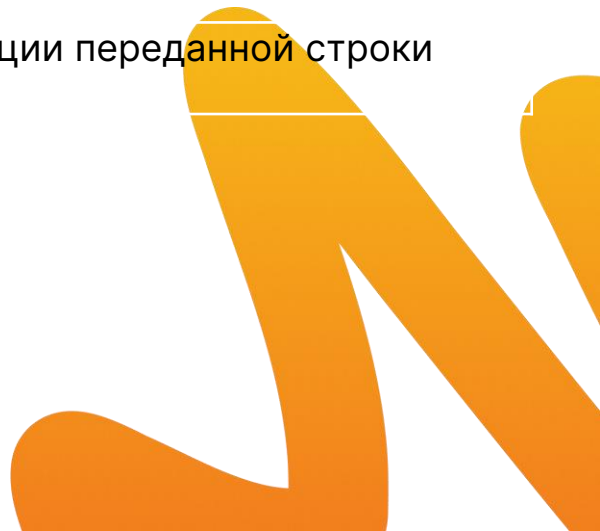
Возвращает поток строк, извлечённых из объекта string, разделённых символом конца строки

**strip(),
stripLeading(),
stripTrailing()**

для удаления начальных и завершающих пробелов из строки

repeat()

Возвращает строку, состоящую из конкатенации переданной строки заданное количество раз



Братья и сестры

Класс StringBuilder



Нельзя использовать сложение строк в циклах! Иначе все промежуточные результаты будут попадать в пул строк и тратить оперативную память.



StringBuilder в Java представляет изменяемую последовательность СИМВОЛОВ.



When I want
to change
value of a String

When I want
to change value
of a StringBuilder

Братья и сестры

Класс **StringBuilder**

Конструкторы:

<code>StringBuilder()</code>	Создает экземпляр без символов и начальной емкостью 16 символов.
<code>StringBuilder(int capacity)</code>	Создает экземпляр без символов и начальной емкости, заданной аргументом <code>capacity</code> .
<code>StringBuilder(CharSequence seq)</code>	Создает экземпляр, содержащий те же символы, что и указанный <code>CharSequence</code> .
<code>StringBuilder(String str)</code>	Создает экземпляр, инициализированный содержимым указанной строки.



Братья и сестры StringBuilder - методы

- **append()** - метод добавляет строковое представление к последовательности.
- **int capacity()** - метод возвращает текущую емкость.
- **char charAt(int index)** - метод возвращает значение char в этой последовательности по указанному индексу.
- **delete(int start, int end)** - метод удаляет символы в подстроке этой последовательности.
- **deleteCharAt(int index)** - метод удаляет символ в указанной позиции в этой последовательности.
- **int length()** - метод возвращает длину (количество символов).
- **replace(int start, int end, String str)** - метод заменяет символы в подстроке этой последовательности, символами в указанной строке String str.
- **reverse()** - метод приводит к замене этой последовательности символов обратной последовательностью.
- **substring()** - метод возвращает новый объект String.
- **toString()** - метод возвращает строку, представляющую данные в этой последовательности.

Задание

Напишите программу «Генератор паролей».

Программа генерирует случайные пароли заданной длины, используя буквы, числа и один из спецсимволов (!, #, %, _).



Процедуры и функции (методы)

Функции и процедуры – это кусочки программы, которые могут повторяться в тех местах, где их вызывают.

Например, `System.out.println();` – это вызов процедуры `println()` объекта `out` (`java.io.PrintStream out` – это поток, выводящий информацию в консоль);

Процедуры и функции в Java называются **методами (method)**.

Процедура отличается от функции тем, что ничего не возвращает (возвращает `void`), поэтому её можно сравнить с пистолетом: вылетевшая пуля сама по себе нас не интересует. Интересует результат работы пистолета.

Функция принимает на вход параметры и возвращает какой-то ощутимый результат, поэтому функцию можно сравнить с мясорубкой, т.к. результат её работы нам нужен для наших кулинарных целей.



Свой метод с блэкджеком

Из чего состоит метод

```
public static void main(String[] args) {  
    //некий код  
}
```

Модификатор доступа (может другой класс вызвать этот метод или нет). Может быть public/protected/private или отсутствовать.

** Модификаторы изучим позже при изучении классов. Пока пишем private*

Примеры:

```
private static void meth1(String[] args) {  
    //некий код  
}
```

```
static void meth2(String[] args) {  
    //некий код  
}
```

Свой метод с блэкджеком

Из чего состоит метод

```
public static void main(String[] args) {  
    //некий код  
}
```

static – если присутствует, то метод относится к классу, если отсутствует – к объекту.

Вызов static-метода: `ClassName.meth1(strArray)` ;

Вызов обычного метода: `objectName.meth2(strArray)` ;

** Пока пишем static в своих методах.*

Примеры:

```
private static void meth1(String[] args) {  
    //некий код  
}
```

```
private void meth2(String[] args) {  
    //некий код  
}
```

Свой метод с блэкджеком

Из чего состоит метод

```
public static void main(String[] args) {  
    //некий код  
}
```

Возвращаемый тип – что получается на выходе в месте вызова метода. Когда метод завершает свою работу, то в месте вызова вместо метода получается возвращаемое значение.

Может быть любой тип или *void*. *void* – ничего не возвращается (метод является процедурой)

Пример вызова метода

```
public static void main(String[] args) {  
    int i = meth2("my String");  
}
```

Примеры:

```
private static void meth1(String arg) {  
    //некий код  
}
```

```
private static int meth2(String arg) {  
    //некий код  
}
```

Свой метод с блэкджеком

Из чего состоит метод

```
public static void main(String[] args) {  
    //некий код  
}
```

Имя метода. Может быть любым словом без пробелов. Не может начинаться с точки, цифры и некоторых символов.

Пишется с маленькой буквы. Чаще всего начинается с глагола (get, set, print, check) или is/has, если метод возвращает boolean.

Примеры:

```
public int getAge() {  
    return age;  
}
```

```
public void setAge(int age) {  
    //некий код  
}
```

Свой метод с блэкджеком

Из чего состоит метод

```
public static void main(String[] args) {  
    //некий код  
}
```

Аргументы (параметры) – данные, передаваемые в метод. Могут быть любых типов. Их может быть сколько угодно. Лучше не делать больше 5 аргументов, т. к. это плохо читается в коде.

Примеры:

```
private String meth1(String str) {  
    return str += " ";  
}
```

```
private String meth2(int number, String str) {  
    return str + number;  
}
```

Свой метод с блэкджеком

Из чего состоит метод

```
public static void main(String[] args) {  
    //некий код  
}
```

Имя метода и входные параметры вместе называются **Сигнатура метода**.

В одном классе можно определить несколько методов с одним названием, но разным набором входных параметров. Это называется **перегрузкой методов**.

Возвращаемый методом тип не является частью примеров:
сигнатуры метода и не считается отличием двух методов.

Пример вызова методов

```
public static void main(String[] args) {  
    String a = addSuffix("my String");  
    String b = addSuffix(10, "my String");  
}
```

```
private static String addSuffix(String str) {  
    return str += " ";  
}
```

```
private static String addSuffix(int number, String str) {  
    return str + number;  
}
```

Свой метод с блэкджеком

Из чего состоит метод

```
public static void main(String[] args) {  
    // тело метода  
}
```

Между { } пишут код, который будет выполняться. Он называется телом метода. Тело метода – это блок кода, то есть переменные, созданные внутри метода, будут доступны только в нём.

Примеры:

```
private static String addSuffix(String str) {  
    return str += " ";  
}
```

```
private static String addSuffix(int number, String str) {  
    return str + number;  
}
```

Свой метод с блэкджеком

Из чего состоит метод

```
public static void main(String[] args) {  
    // тело метода  
    return;  
}
```

return немедленно прекращает выполнение метода и возвращает выполнение программы в точку вызова метода. *return* в методе *main* завершает программу.

Если возвращаемый тип метода не *void*, то после *return* указывают возвращаемый результат.

Если возвращаемый тип метода *void*, то *return* можно не указывать.

Один и тот же метод можно закончить в разных точках, указав *return* для соответствующих условий завершения.

Примеры:

```
void meth1() {  
    //некий код  
    return;  
}
```

```
String meth2() {  
    return "some string";  
}
```


Свой метод с блэкджеком

Перехват управления

В программе в точке вызова метода в метод передаётся поток управления и входные параметры (при наличии).

Когда метод завершает работу, то управление переходит в точку вызова метода. Туда же передаётся значение, указанное после *return*.



Свой метод с блэкджеком

Рекурсия

Рекурсия - это вызов метода из самого себя. Такой способ позволяет заменить циклы. В этом случае код часто выглядит проще, но требует больше ресурсов для выполнения, т.к. каждый вызов метода резервирует часть оперативной памяти.

В строке

```
return addSame(str + "_" + str, length);
```

Метод вызывает сам себя, передавая изменённое значение параметров.

```
public class Recursion {  
    public static void main(String[] args) {  
        String startStr = "ABC";  
        int minLength = 15;  
        String result = addSame(startStr, minLength);  
        System.out.println(result);  
    }  
  
    private static String addSame(String str, int length) {  
        System.out.println(str);  
        if (str.length() > length) {  
            return str;  
        }  
        return addSame(str + "_" + str, length);  
    }  
}
```

Задание

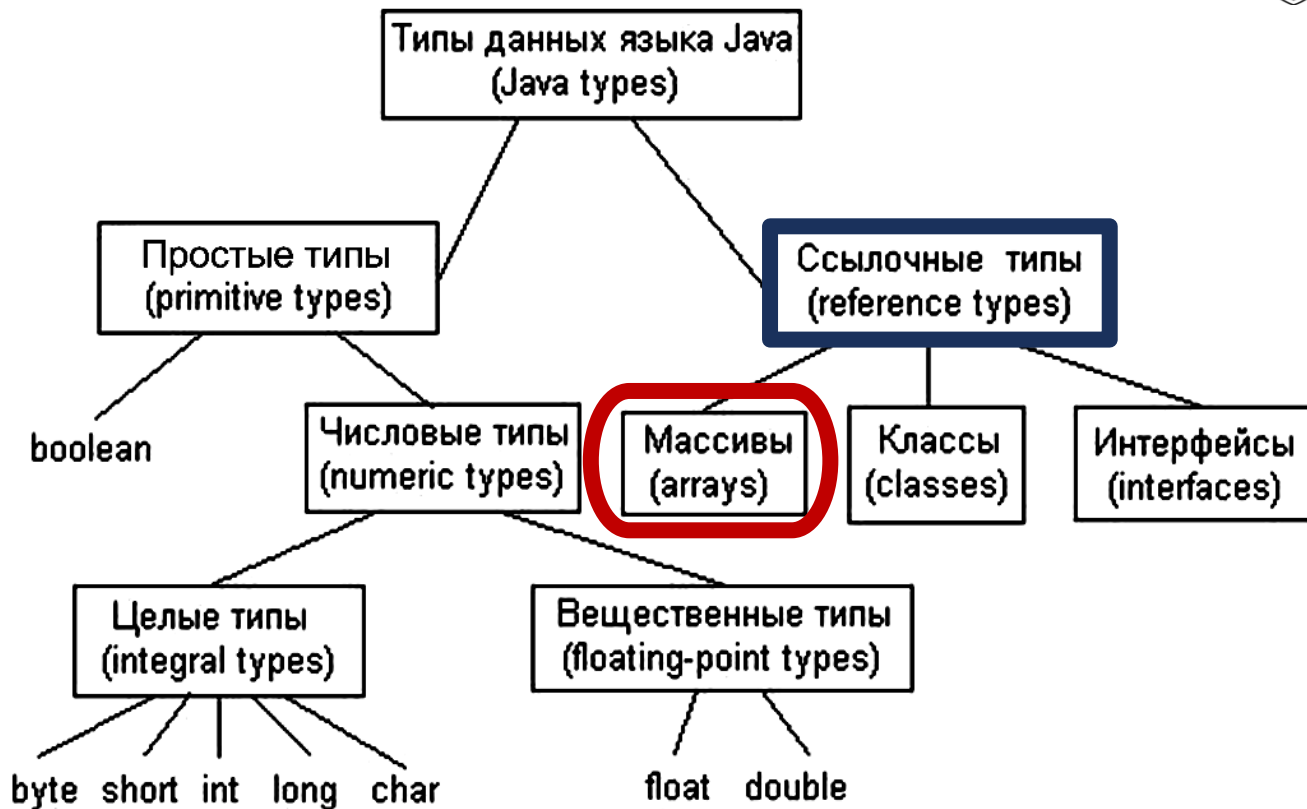
Напишите метод, который принимает 2 строки и объединяет их (конкатенация). Вызовите метод в программе. Результат выведите в консоль.

Сделайте перегрузку предыдущего метода, чтобы он принимал 3 строки. Для склеивания двух строк вызовите предыдущий метод, затем добавьте третью строку.

Сделайте перегрузку предыдущего метода, чтобы он принимал 4 строки. Вызовите метод в программе. Результат выведите в консоль.



Решение

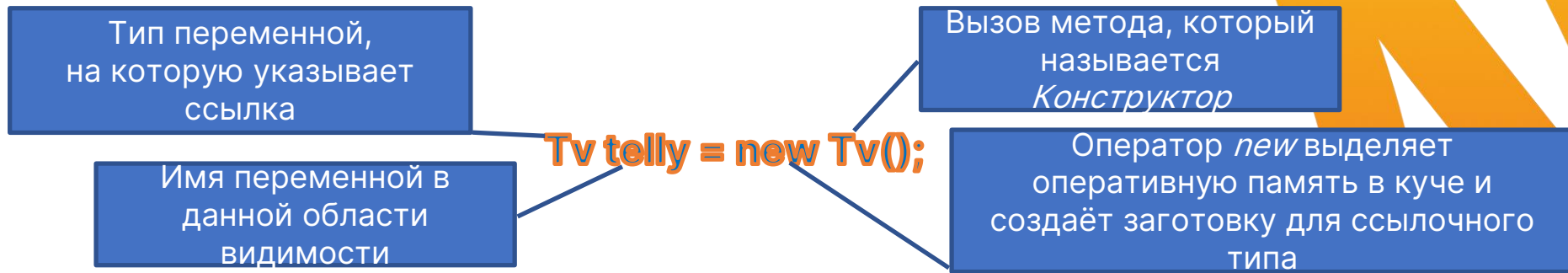


В ссылке

Ссылочные типы данных Java

Ссылочный тип данных – это тип данных,

- экземпляры которого лежат в *куче* (heap, общая оперативная память), а ссылки на начало объекта хранятся в нашей программе (в области памяти под названием *стэк*). Ссылка – это адрес ячейки, «визитная» карточка объекта. Программа по ссылке находит весь объект и может с ним что-то делать;
- который составлен из примитивных типов данных и ссылок на другие ссылочные типы (как матрёшка);
- передаётся в методы по ссылке (примитивные типы передаются по значению);
- новые объекты создаются через оператор *new*. Например,



В ссылке

Куча и стек

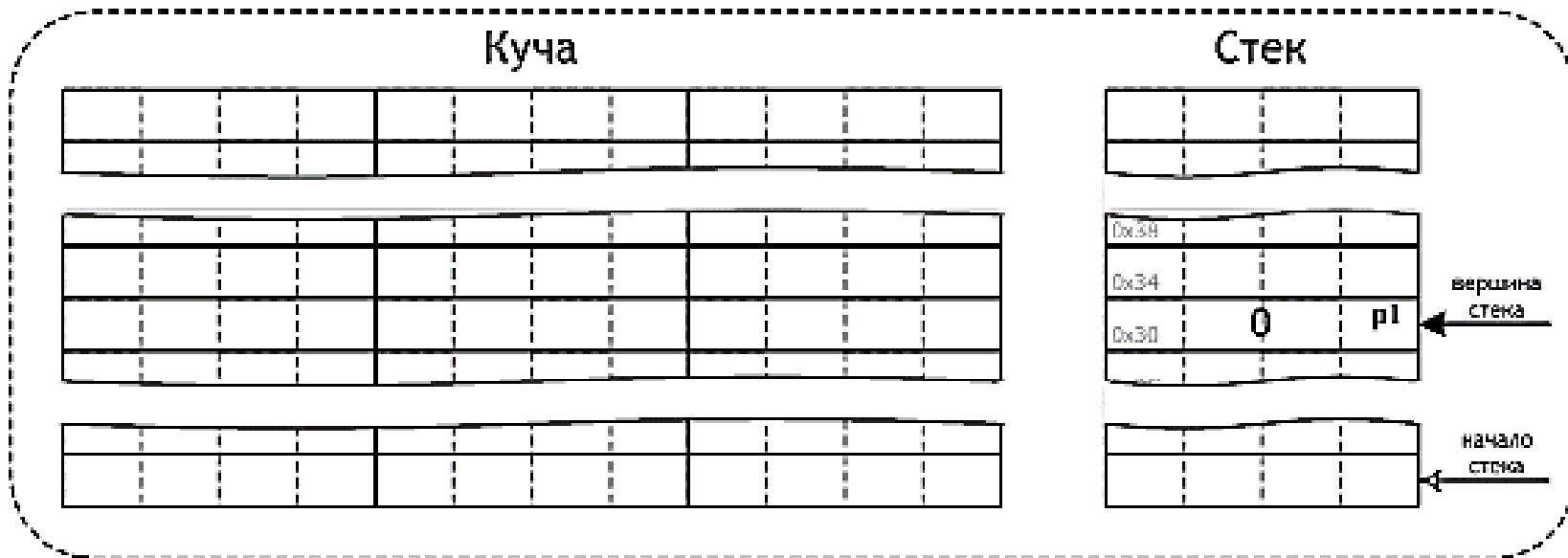


В ссылке

Куча и стек

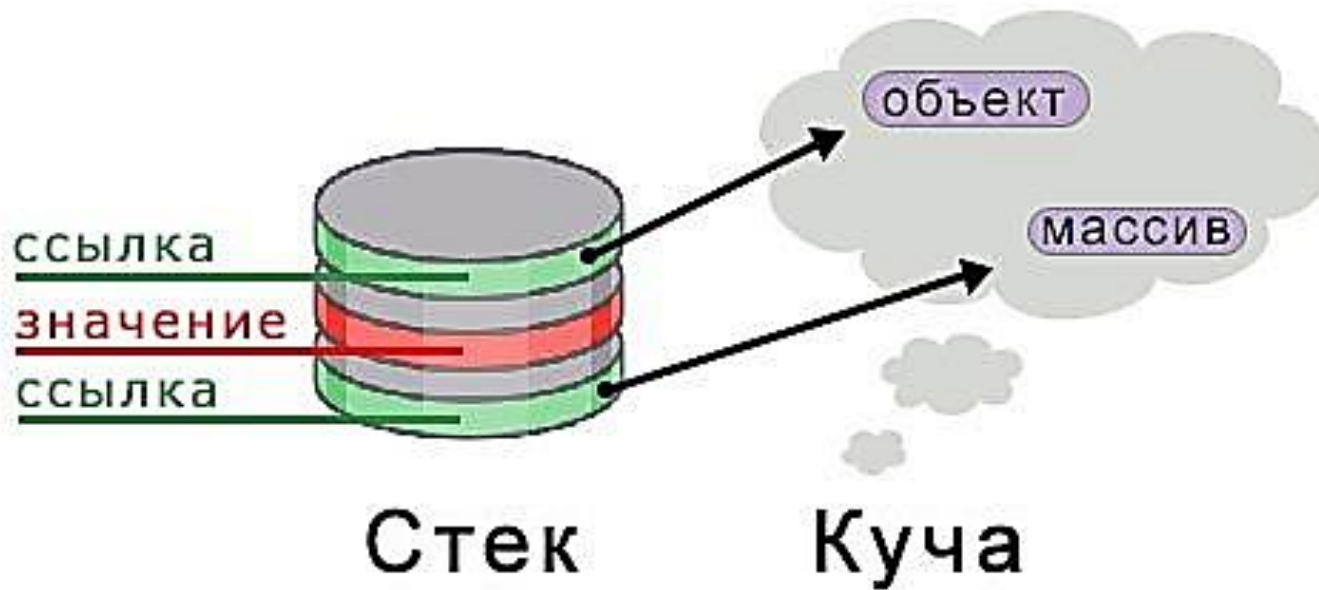
Куча и стек – это участки оперативной памяти. **Стек** имеет небольшой размер, он упорядочен (все переменные хранятся в ячейках по порядку). **Куча** – вся остальная оперативная память, доступная программе. В куче хранятся большие объекты, которые могут быть разбросаны в разных частях оперативной памяти.

Оперативная память



В ссылке

Куча и стек



Не лесной, не жилой, не горный Массивы

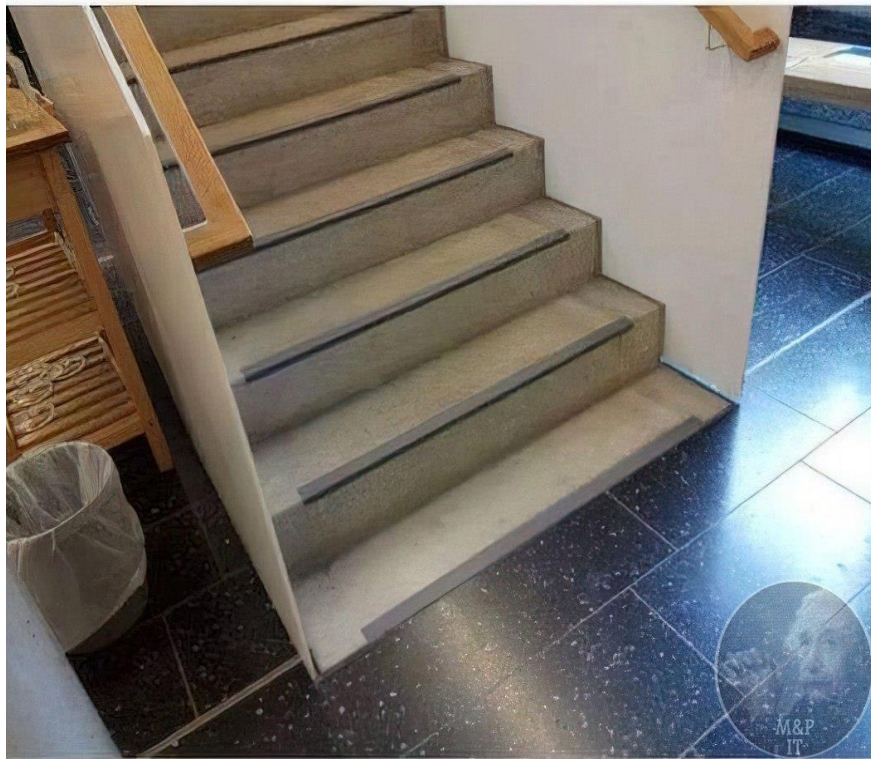
- Переменная массива Java может быть объявлена, как и другие переменные, `<тип>[] <имя>`.
- Переменные в массиве упорядочены, и каждая имеет индекс, начинающийся с 0.
- Размер массива должен быть указан как целое значение.
- Размер массива не может быть изменен (после инициализации).

Index

0	1	2	3	4	5	6
5	0	3	10	0	0	0

Длина = 7

**Ступеньки начинаются с 0 :)
Строитель видимо программист.**



Не лесной, не жилой, не горный

Создание и инициализация массива

```
int[] intArray1 = new int[0];
```

Создан пустой массив

```
int[] intArray2 = new int[] {1, -2, 3};
```

Создан массив из трёх элементов

```
int[] intArray2 = {1, -2, 3};
```

Краткая форма создания массива

```
int elem0 = intArray2[0];
```

Получить значение элемента массива

```
intArray2[0] = 15;
```

Установить значение элемента массива

Не лесной, не жилой, не горный Инициализация массива

Элементы в массиве по умолчанию, будут автоматически инициализированы:

- **0** (ноль, для числовых типов)
- **false** (для логического значения)
- **null** (для ссылочных типов)



Не лесной, не жилой, не горный Доступ к элементам массива

Чтобы перебрать все элементы массива, используется цикл.

```
public static void main(String[] args) {  
    int[] array = {1, 2, 3, 4, 5, 6, 100}; // создаём массив  
    // Способ перебора 1  
    for (int i = 0; i < array.length; i++) {  
        array[i]--; // изменим каждый элемент - уменьшим на 1  
        System.out.println(array[i]); // выводим каждый элемент  
    }  
    // Способ перебора 2  
    for (int j : array) {  
        System.out.println(j); // выводим каждый элемент  
    }  
    // Способ превратить массив в строку  
    String asString = Arrays.toString(array); // класс Arrays содержит множество полезных методов  
    System.out.println(asString);  
}
```

Первый помощник массивов

Класс **Arrays**

У класса `Arrays` есть набор полезных методов, которые облегчают работу с массивами:

- `copyOf` – копирует массив (в новый участок памяти);
- `toString` – собирает данные массива в строку;
- `sort` – сортирует массив;
- `copyRange` – копирует часть массива в новый массив.



Задание

Создайте массив строк и массив чисел. Выведите их в консоль.

Заполните массив чисел случайными значениями.

Заполните массив строк случайными строками (используйте `UUID.randomUUID().toString()`)

Вновь выведите массивы в консоль.

Занулите каждый чётный элемент в массиве чисел.

Выведите массив чисел в консоль.

Отсортируйте массив строк.

Выведите массив строк в консоль.



2

Домашнее задание

Задание

Прочитать эту презентацию и, при необходимости, пересмотреть лекции.

Если какая-то тема требует более глубокого повторения, напишите преподавателю.

Если будут вопросы, напишите их преподавателю.



ЗАКЛЮЧЕНИЕ

