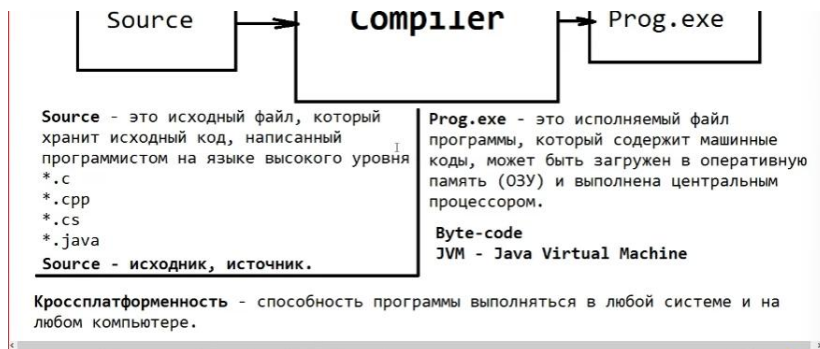


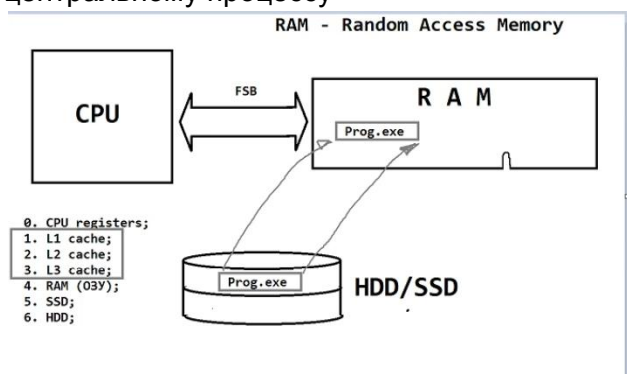
А. Введение в программирование. (27.06.2023)

Инструменты разработчика (Developer Tools)

1. Текстовый редактор с подсветкой синтаксиса notepad++, aditor, source editor, sublime text
2. Compiler - это программа которая преобразует исходный код написанный на языке высокого уровня в машинный код, который может быть загружен в оперативную память и выполнен центральным процессором.



Абсолютно любая программа представляет собой последовательность команд центральному процессу

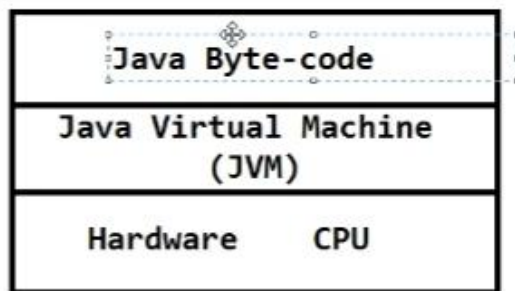


Программы написанные на языке java не преобразуются непосредственно в машинный код и не выполняются непосредственно центральным процессором. Это является преимуществом и недостатком данного языка программирования.

Программы написанные на языке java преобразуются компилятором в так называемый байт код. Этот код очень похож на машинный язык. Он очень напоминает команды центрального процессора, но ни один CPU (central processing unit) не может его выполнить. Byte code может быть выполнен только виртуальной машиной JVM (Java Virtual Machine). Преимуществом данного подхода является кроссплатформенность. Кроссплатформенность - способность программы выполняться в любой системе и на любом компьютере. Программы написанные на java работают везде где работает виртуальная машина java (JVM). Программисту не нужно думать о том, в какую процессорную архитектуру скомпилировать этот свой исходный код. Это берет на себя виртуальная машина JVM. Благодаря этому java долгое время монополист в кроссплатформенной разработке.

Архитектура - это свойства центрального процессора. Архитектура - это набор команд понятный центральному процессору которую он может прочитать и выполнить.

И проблема состоит в том, что существует множество разных процессоров от разных производителей, каждый из которых имеет свою архитектуру. То есть свой понятный набор команд, которую он может выполнять. Наиболее распр архитектурами явл x86, ARM, Power-pc, PC-98, Ultra Spark....



Недостатком данного подхода является низкая производительность, а также невозможность работать напрямую с аппаратным обеспечением компа. JVM полностью абстрагирует нас от аппаратного обеспечения и забирает на себя часть производительности физической машины. Именно поэтому на языке java никогда не пишут драйверы, ОС и высоко производительные игры. Именно в этих отраслях нужен прямой доступ к аппаратному обеспечению и максимальная производительность. драйверы, ОС и высокопроизводительные игры пишутся на языках C и C++. А некоторые критические секции могут быть написаны на ассемблере. Ассемблер - это машинный язык. Но java отлично подходит для написания пользовательских приложений.

Application Development:	WindowsDev/Linux-UNIX Dev/AndroidDev/GameDev
Enterprise Development:	Серверные приложения для предприятий
Web-Development:	Разработка Web-сайтов (FrontEnd/BackEnd)
Embedded Development:	Разработка встраиваемых систем
(Firmware - Прошивка)	(микроконтроллеры, специализированные компьютеры, бытовая техника.....)

3. Отладчик (debugger) - это программа, которая позволяет пошагово выполнить нашу программу для того чтобы найти в ней ошибки на этапе выполнения или ошибки в алгоритме нашей программы.

Есть три типа ошибок (Error types)

- ☐ Compile error - ошибка на этапе компиляции. Прежде чем преобразовывать исходный код в машинный код компилятор проверяет его на наличие синтаксических ошибок. Например в отсутствии скобки, точки запятой, необъявленная переменная. Они проявляются на этапе компиляции. С этими ошибками работать проще всего. Они легче всего находятся. Компилятор как правило выдает сообщение об ошибке с указанием номера строки в которой допущена эта ошибка. Также у каждой ошибки есть свой код. По которому достаточно легко найти описание ошибки в документации либо же в интернете.
- ☐ Link error - ошибки на этапе компоновки.
- ☐ Runtime error - ошибки на этапе выполнения. Возникает когда программа запустилась и работает, но аварийно завершила свою работу и выдает неправильный результат. Как раз для поиска таких ошибок нужен отладчик.

4. Система контроля версий (VCS - Version control system) позволяет в любой момент вернуться на промежуточную версию, а также хранить код на централизованном сервере и организовать командную работу над проектом
5. Справочная система и документация. может быть установлена как на локальном компьютере но на сегодняшний день большинство документации находятся в онлайн ресурсах

Все эти инструменты объединены IDE (Integrated development environment) интегрированная среда обработки

Существует множество разных IDE таких как NETBEANS, Eclipse, Visual studio, visual studio code, jetbrains intellij ideA.

IntelliJ IDEA

Вы просто начинаете писать какое-нибудь слово, а она тут же вам предлагает подсказки, чтобы его закончить. Одна из самых часто используемых клавиш в автодополнении — это клавиша табуляции «Tab».

Например, вы хотите написать System. Для этого вам нужно написать Sys и нажать кнопку «Tab»: остальное IDEA закончит за вас.

Например, вы можете написать public static void main(String[] args) меньше чем за секунду ? Для этого нужно написать **4 буквы psvm** и нажать Tab. А IDEA заменит «psvm» на «*public static void main(String[] args)*».

Также есть способ быстро написать *System.out.println()*; Для этого тоже нужно написать **4 буквы «sout»** и нажать Tab



Быстро написать цикл в IDEA можно с помощью команды **fori + Tab**. При этом IDEA заменит fori на код:

```
for (int i = 0; i < ; i++) {  
}
```

Оборачивание: Ctrl+Alt+T Иногда в работе программиста возникают ситуации, когда хочется что-то сделать с уже написанным кодом. Для этого в IntelliJ IDEA есть специальная команда, которая позволяет «обернуть» написанный код чем-то еще.

Стиль кода: Ctrl+Alt+L И еще один полезный момент. Очень часто в результате копирования кода его выравнивание нарушается: где-то лишние пробелы, где-то их не хватает и т.д. Вроде бы и работающий код, а выглядит как черт знает что.

Отладка программы

Режимы запуска программы	Иконка на панели	Горячие клавиши
Обычное выполнение		Shift+F10
Запуск в режиме отладки		Shift+F9

Пошаговое выполнение

Если ваша программа работает в режиме отладки, вы также можете выполнять ее пошагово: один шаг — одна строчка. Для пошагового выполнения есть две горячие клавиши: F7 и F8

Пошаговое выполнение с заходом в методы

Если у вас есть в программе собственные методы, и вы хотите, чтобы в режиме отладки ваша программа не просто выполнялась пошагово, но и заходила внутрь ваших методов, для «захода в метод» вам нужно нажимать не клавишу F8, а клавишу F7.

Все очень просто. Если вам не сильно важно, что и как происходит внутри метода, вы нажимаете F8, если важно — F7 и пошагово выполняете весь его код.

Изменение значений переменной

Кстати, если вы хотите протестировать, как ваша программа поведет себя при определенных значениях переменных, вы можете просто поменять значения любых переменных прямо во время работы программы (в режиме отладки). Для этого нужно кликнуть правой кнопкой мышки на имени переменной или нажать F2:

Выполнение фрагмента кода

Также в любой момент работы программы вы можете выполнить произвольный код. Это делается с помощью комбинации кнопок Alt+F8 или пункта контекстного меню:

GK__Java_urok_01

Основной целью технологии Java является принцип – «Написанное один раз – работает везде» написав один раз код приложения, можно повторно его системах, без каких-либо дополнительных усилий.

Java платформа – это набор средств для разработки на языке Java, которые поставляются разработчику в виде единого готового решения. В платформу входят:

- стандартные библиотеки;
- средств компиляции;
- средства выполнения кода;
- средства подготовки документации по коду;
- примеры;
- другие средства для разработки.

Виртуальная машина (JVM) – это спецификация, описывающая абстрактную машину, в которой могут выполняться приложения на Java. Машина считается виртуальной, так как большую ее часть составляют различные программные компоненты.

Компилятор – это средство, которое преобразует исходный код в байт-код. Java компилятор написан на языке Java.

Байт код – это набор инструкций на промежуточном языке, предназначенный для выполнения виртуальной машиной. Каждая инструкция кодируется одним байтом.

В Java контроль за использованием памяти возложен на специальный механизм, который называется сборкой мусора (garbage collector). Наличие механизма автоматической сборки мусора (более не используемых данных) упрощает процесс разработки и уменьшает количество ошибок при работе с памятью.

В. Работа в IDE (29.06.2023)

В любой IDE для того чтобы написать программу нужно создать проект. И добавить туда исходных файл.

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

Java - это ООП (объектно ориентр программирование), в объектно ориентир языках абсолютно все делает при помощи классов.

1. В любой программе есть **главный класс** Main `public class Main { }`

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

2. main(String[] args) - **это главная функция (метод)** которая является точкой входа в программу поскольку с нее начинается выполнение любой программы.
3. String[] args - **это принимаемый параметр функции** или же метода **main**. Через эту строку мы можем передавать исходные данные для работы нашей программы. Это означает, что метод `main()` получает на входе массив строк
4. Функция может не принимать никаких параметров, в таком случае `()` оставляют пустыми, но их обязательно нужно писать, поскольку они отличают функцию от переменной.
5. **Ключевое слово `void`** или же пустота показывает, что функция (метод) `main` ничего не возвращает по завершению.
6. **`Public` показывает то, что функция `main` явл открытой** и может быть вызвана из любого места программы.
7. **`static` показывает, что метод `main` является статическим**, т.е. его можно вызвать не создавая объект.
8. **`{ }` показывают начало и конец функции `main`**. Они определяют пространство имен и область видимости функции `main`.

```
System.out.print("Hello world!");
```
9. **`System.out.print` метод `print` класса `System` выводит данные на экран**. Он расположен в библиотеке классов в `JDK`.
10. методу `print` передаем строку которую надо вывести на экран `("Hello world!")`
11. `("Hello world!")` - **это строковая константа**. Строковая константа это последовательность произвольных символов. заключенная в двойные кавычки `"Hello world!"`.
12. **`;` - показывает конец выражения на языке `java`**. Выражение это синтаксическая конструкция заканчивающаяся `;`

Классы и объекты будут подробно рассмотрены на теме ооп.

C. Escape - последовательности (Esc)

Esc нужны для выравнивания в консоль информации. а также для вывода некоторых спецсимволов. Esc последовательности одиночные символы, экранированные символом `'\'`

`'/'` slash - косая черта, дробная черта

`'\'` backslash обратная косая черта

Экранированный символ значит, что перед этим символом стоит `'\'` например `'\n'` символ `'n'` экранирован символом `'\'`

`println` и `\n` переводит текст на следу строчку, но написав много раз `n` можно переводить на несколько раз.

`\t\t\t` похоже на пробел, смещает слово на указанное кол-во `t` (1 `t` до 8 симв (пробелов)).

Сам `'\'` **это символ отмены специального значения другого символа**. То есть он отменяет специального значения символа следующего за ним. Например

`""world""\n` если поставить `\"` `\"world""\n` из за этого они перестали быть закрывающим символом, они стали обычными кавычками. `\"` отменяет спец значение кавычек и позволят вывести кавычку в консоль. `\"` отменяет специальное значение в том числе и самого себя.

Некоторым символам `\"` наоборот придает специальное значение. Например `\\n` new line переводит курсор в начало новой строки

`\t` (tab) выводит на экран табуляцию. 1 символ до 8 пробелов.

Следующий оператор присваивает переменной `ch` символ табуляции: `ch = '\t';`

А так переменной `ch` можно присвоить символ одинарной кавычки: `ch = '\'' ;`

Управляющая последовательность	Описание
\'	Одинарная кавычка
\"	Двойная кавычка
\\	Обратная косая черта
\r	Возврат каретки
\n	Новая строка (также известная как перевод строки)
\f	Подача страницы
\t	Табуляция
\b	Забой
\ddd	Восьмеричный символ (ddd – восьмеричная константа)
\uxxxx	Шестнадцатеричный символ Unicode (xxxx – шестнадцатеричная константа)
\s	Пробел (последовательность добавлена в JDK 15)
\конец-строки	Строка продолжения (применяется только к текстовым блокам; последовательность добавлена в JDK 15)

```
// Демонстрация использования управляющих последовательностей в строках
class StrDemo {
public static void main(String[] args) {
System.out.println("Первая строка \nВторая строка"); //Использовать
последовательность \n для вставки символа новой строки
System.out.println("A\tB\tC");
System.out.println("D\tE\tF"); t}
}
```

Комментарии - это фрагменты исходного кода, которые игнорируются компилятором.

Комментарии используют в двух случаях. 1) для комментирования своих действий в коде. 2) для временного отключения некоторых фрагментов кода без удаления это кода.

Комментарии бывают строчные и блочные.

Строчный комментарий - после // игнорируются все символы строки до конца строки.

Блочный комментарий - /* */ у которого есть начало и конец и он позволяет закомментировать несколько строк кода или же часть строки кода.

Д/з В новом проекте используя вывод на экран и Escape - последовательности вывести в консоль стишок в “лесу родилась елочка”. Стишок должен быть выровнен по середине окна консоли, а его куплеты должны разделяться пустой строкой. Прислать java файл. Edit → Find → Replace найти и заменить на... Replace all

GK__Java_urok_02

Примитивные типы данных (базовые типы) – это типы данных присутствующие в синтаксисе языка. Для использования примитивных типов нет необходимости создавать свои классы или использовать какие-либо библиотеки.

Примитивные типы данных делятся на:

целочисленные примитивные типы данных могут хранить в себе только целые числа (без дробной части), из диапазона заданного размерностью типа.

дробные типы данных предназначены для хранения значений дроби. Дробный тип данных может быть представлен в виде десятичной дроби.

Ссылочный тип (объектный тип) – это тип который в качестве значения содержит ссылку на объект.

Символьный тип может хранить в себе только целые числа, которые интерпретируются, как коды символов из таблицы на основе юникода (Unicode). Юникод – это стандарт кодирования символов, где каждому символу соответствует определенный код. Если нет возможности набрать в коде символ с клавиатуры он может быть заменен на запись символа в виде юникода. Например: '\u0056'

Логический тип используется для хранения значений, полученных в результате вычисления логических выражений, или может задаваться логическими литералами: где true – обозначает истину, false – означает ложь.

Таблица неявных преобразований

Исходный тип	Типы, в которые можно преобразовать неявно	Потеря точности
byte	short, int, long	Нет
short	int, long	Нет
int	long	Нет
int	float, double	Да
char	int	Нет
long	float, double	Да
float	double	Нет

Синтаксис явного преобразования: <тип> i = (<тип>) значение исходного типа;
int имя;

Краткая запись создания переменных int имя1, имя2, имя3;

Важно!!! При явном преобразовании типов, контроль за переполнением значения или потери точности полностью ложится на плечи программиста.

Пример сложения с использованием объектов:

```
Integer sum = new Integer(2) + new Integer(2);
```

Пример сложения с примитивными типами: int sum = 2 + 2;

Все классы обертки являются неизменными (нельзя изменить значение хранящееся внутри объекта класса-обертки), так же нельзя наследоваться от классов-обертки.

Примитивный тип	Тип-оболочка
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Присваивание значений. Чтобы занести значение в переменную типа int, нужно воспользоваться командой: имя = значение; или можно так int имя = значение;

```
/*
Программа для преобразования галлонов в литры
*/
class GalToLit {
public static void main(String[] args) {
double gallons; // хранит количество галлонов
double liters; // хранит результат преобразования в литры
gallons = 10; // начать с 10 галлонов
liters = gallons * 3.7854; // преобразование в литры
System.out.println(gallons + " галлонов соответствует " + liters + " литрам.");}
}
```

```
/*
Вычисляет количество кубических дюймов в кубической миле.
*/
class Inches {
public static void main(String[] args) {
long ci;
long im;
im = 5280 * 12;
ci = im * im * im;
System.out.println("В кубической миле содержится " + ci +
" кубических дюймов.");}
}
```

В методе main объявите переменные intValue, numericValue, size, number типа int. Присвой им разные значения. Используй одну команду для создания и инициализации переменных. Значениями могут быть любые целые числа.

```
public class Solution {
    public static void main(String[] args) {
        int intValue= 4, numericValue = 5, size = 8, number = 10;    }
```

В переменной centimeters записано количество сантиметров. В переменную meters нужно записать количество полных метров в centimeters (1 метр = 100 см). Для вычисления используя переменную centimeters и оператор деления. Для объявления и инициализации meters используй одну команду.


```
public class Solution {
    public static void main(String[] args) {
        int centimeters = 243;
        int meters = centimeters/100;
        System.out.println(meters); }
}
```

Остаток от деления целых чисел

Это очень полезный оператор, и используется он довольно часто. Например, чтобы узнать, четное число или нет, достаточно поделить его на 2 и полученный остаток сравнить с нулем. Если остаток от деления равен нулю, число четное, если равен единице — нечетное. Выглядит эта проверка так: $(a \% 2) == 0$

В переменной number записано число. В переменную lastDigit нужно записать последнюю цифру этого числа. Для вычисления используйте переменную number и оператор «остаток от деления». Для объявления и инициализации lastDigit используйте одну команду.

```
public class Solution {
    public static void main(String[] args) {
        int number = 546;
        int lastDigit = number % 10;
        System.out.println(lastDigit); }
}
```

Все целочисленные типы-обертки наследуются от абстрактного класса Number. Значения передаются в класс-обертку с помощью конструктора, в виде литерала или переменной примитивного типа или в качестве строки.

Классы-обертки имеют статические и нестатические методы.

Статические методы: методы вида parseXXX преобразуют строку в значение примитивного типа.

Метод valueOf преобразует строку или число в класс-обертку.

Метод toBinaryString есть только у классов-обертки целых типов преобразует число в строку в бинарном представлении.

Метод toHexString преобразует число в строку в шестнадцатеричном представлении.

Совет. Старайтесь всегда использовать примитивные типы, а обертки используйте только там, где невозможно использовать примитивные типы.

Автоматическая упаковка (autoboxing) – это механизм неявного создания (без использования оператора new) объекта класса-обертки соответствующего примитивного типа.

Автоматическая распаковка (unboxing) – это механизм неявного преобразования объекта типа-обертки в примитивный тип соответствующего типа.

// (CTRL+ /) – строчный комментарий, все что находится после двойных обратных черт до конца строки не будет скомпилировано

Блочный комментарий (CTRL + Shift + /) позволяет выделить часть исходного кода, которая будет исключена из компиляции:

/* — начало блочного комментария,

*/ — конец блочного комментария.

Переменная – это именованная область памяти, в которую может быть записано или перезаписано и откуда может быть прочитано значение определенного типа.

Переменные объявленные внутри метода называются **локальными переменными** и перед непосредственным использованием требуют явной инициализации.

Переменная объявленная внутри тела класса называется **полем класса**, в случае отсутствия инициализации переменная экземпляра принимает значение по умолчанию (в зависимости от типа).

```
class A
{
    static int x; // поле класса
    public static void main(String[] args) {
        int y; // локальная переменная
        System.out.println(y); // результат ошибка
        // компиляции с сообщением
        System.out.println(x); // результат 0
    }
}
```

Константа – это переменная которая должна быть инициализирована в месте объявления (или в конструкторе класса) и не может изменять своего значения на протяжении видимости этой переменной в коде. Для обозначения константы используется ключевое слово **final**.

Литерал – это явно заданное в коде значение определенного типа. Для задание литералов в коде программы используется специальный синтаксис, см. таблицу.

Тип литерала	Тип данных	Спец-символы	Пример
Целочисленный	int	x	3, 03, 0x3
	long	l, L	3l, 3L
Дробный	float	f, F, e	1.5f, 1.5F, 1.5e-1f
	double	d, D, e	1.5, 1.5d, 1.5D, 1.5e-1
Булевый	boolean	true, false	true, false
Символьный	char	'	'a', '\u0041'
Ссылочный	Все ссылочные	null	null

Для вывода данных на консоль необходимо использовать **класс System** и его статическую переменную out.

System.out.println(«text»); // вывод текста с переходом на новую строку

System.out.println(5); // вывод литерала с переходом на новую строку

int i = 6;

System.out.println(i); // вывод значения переменной

System.out.print(«text»); // вывод текста без перехода на новую строку

System.out.printf("x = %f", 0.5f); // форматированный вывод переменных

Для ввода данных с консоли необходимо использовать класс Java.util.Scanner. Класс Scanner имеет ряд методов, позволяющих работать с потоком ввода.

Класс Scanner представляет из себя поток данных, получаемых из источника.

Когда используется класса из библиотеки или из другого пакета необходимо указать, откуда его загружать. Для этого в начале файла до тела класса необходимо добавить:

```
import java.util.Scanner;
```

Пример программы, которая считывает с клавиатуры два числа и выводит их сумму:

```
import java.util.Scanner;
public class Solution {
    public static void main(String[] args)
    {
        Scanner console = new Scanner(System.in);
        int a = console.nextInt();
        int b = console.nextInt();
        System.out.println(a + b);
    }
}
```

Считай с клавиатуры три строки. А затем: Выведи на экран третью строку в неизменном виде. Выведи на экран вторую строку, предварительно преобразовав ее к верхнему регистру. Выведи на экран первую строку, предварительно преобразовав ее к нижнему регистру.

```
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    String println1 = console.nextLine();
    String println2 = console.nextLine();
    String println3 = console.nextLine();
    System.out.println(println3);
    System.out.println(println2.toUpperCase());
    System.out.println(println1.toLowerCase());
}
```

Считай с клавиатуры три целых числа. Выведи на экран их среднее арифметическое.

```
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    int a = console.nextInt();
    int b = console.nextInt();
    int c = console.nextInt();
    System.out.println((a + b + c)/3);
}
```

```

public static void main(String[] args)
{
    // создаем объект класса сканер и передаем в конструктор стандартный поток
    // вывода
    Scanner scanner = new Scanner(System.in);
    System.out.println("Введите число");
    // возвращает истину, если введен целое число
    boolean isInt = scanner.hasNextInt();
    System.out.println(isInt);
    // проверяем, являлось ли введенное значение целым числом
    if (isInt)
    {
        // считываем из консоли значение целого числа в переименую x
        int x = scanner.nextInt();
        System.out.println("Вы ввели число « + x");
    }
    else
    {
        System.out.println("Это не целое число");
    }
}

```

Ввод данных из строки

Класс Scanner умеет считывать данные из разных источников. И один из этих источников — строка текста.

```

String str = "текст";
Scanner scanner = new Scanner(str)

```

Вместо объекта System.in мы при создании объекта типа Scanner передаем в него строку — str. И теперь объект scanner будет считывать данные из строки

```

import java.util.Scanner;
public class Solution {
    public static void main(String[] args)
    {
        String str = "10 20 40 60";
        Scanner scanner = new Scanner(str);
        int a = scanner.nextInt();
        int b = scanner.nextInt();

        System.out.println(a + b);
    }
}

```

Из статьи: <https://journal.top-academy.ru/ru/main/library/page/index/8>

Программисты Java знают, что в языке имеются две стандартные возможности для работы с пользовательским интерфейсом — AWT и Swing. AWT — это платформо-зависимая реализация графического интерфейса пользователя. Скорость работы большинства ее реализаций вполне удовлетворительна, но количество функций весьма ограничено. В качестве альтернативы AWT разработана библиотека Swing. Она целиком основана на возможностях языка, имеет множество функций и платформо-независимый, но скорость ее работы невысока.

К сказанному можно добавить, что в Java «встроена» поддержка мультизадачности. При создании многопоточковых приложений разработчику, как правило, приходится использовать средства операционной системы (семафоры, мьютексы) для

синхронизации задач. Java же предлагает универсальное решение на основе конструкций самого языка.

Резюмируя, можно сказать, что по сложности программирования Java и в сравнение не идет с Си++ или Паскалем. Развитые возможности языка, поддержка “сбора мусора”, единая стандартная библиотека классов, контроль со стороны компилятора — все это заметно упрощает создание приложений и ускоряет их отладку.

Основной конек Java — платформенезависимость, т. е. независимость программных средств, работающих на виртуальной машине, от аппаратного обеспечения и операционной системы.

Главное, что отличает статическую (static) типизацию от динамической (dynamic) то, что все проверки типов выполняются на этапе компиляции, а не этапе выполнения.

Тип String

String имя; Чтобы занести в переменную типа String значение, нужно воспользоваться командой

имя = "значение";

Все значения типа String представляют собой строки текста и должны быть заключены в двойные кавычки.

В методе main объявите переменные word, phrase, line и text типа String. Присвой им разные значения. Используй одну команду для создания и инициализации переменных. Значениями могут быть любые строки.

```
public class Solution {  
    public static void main(String[] args) {  
        String word = "xsf", phrase = "ert", line = "rt", text = "vv"; }  
}
```

Конкатенация — склеивание строк или сложение строк.

"значение1" + "значение2"

В переменную tagline нужно записать строку "JustDolt". Используй s1, s2, s3 и конкатенацию строк.

```
public static void main(String[] args) {  
    String s1 = "Do";  
    String s2 = "It";  
    String s3 = "Just";  
    String tagline = s3 + s1 + s2;  
    System.out.println(tagline); }  
}
```

Заполни пробелы пустотой... или наоборот. Используй переменную emptiness, строки с одним пробелом и конкатенацию строк, чтобы записать в переменную fullness строку "пустота пустота пустота".

```
public static void main(String[] args) {  
    String emptiness = "пустота";  
    String fullness = emptiness + " " + emptiness + " " + emptiness ;  
    System.out.println(fullness); }  
}
```

Преобразование к строковому типу. Мы складываем тип String с каким-нибудь другим типом. Примеры:

```
int a = 5;  
String name = "Аня" + a; // name содержит строку Аня5
```

```
int a = 5;  
String name = "1" + a; //name содержит строку 15
```

Операция сложения выполняется слева направо, так что результат может быть несколько неожиданным

```
int a = 5;  
String name = a + a + "1" + a; //name содержит строку 1015. Порядок выполнения: ((a + a) + "1") + a
```

Переменной типа String также можно присвоить пустую строку:

```
String anotherMessage = "";
```

Преобразование строки в число. Чтобы преобразовать число в строку в Java, достаточно сложить его и пустую строку:

```
String str = "" + число;
```

А вот что делать, если нужно преобразовать строку в число? Ну, любую строку в число преобразовать нельзя. Но вот если строка состоит только из цифр, то все-таки можно. Для этого есть специальный метод у класса Integer. Выглядит эта команда так:

```
int x = Integer.parseInt(строка)
```

Где int x – это объявление целочисленной переменной x, а строка — это число заданное в виде строки (строка состоящая из цифр).

```
String str = "123";  
int number = Integer.parseInt(str); // number содержит число 123;
```

В переменную digits нужно записать строку "60". Используй переменные x, y, z, пустую строку и конкатенацию строк.

```
public static void main(String[] args) {  
    int x = 2;  
    int y = 4;  
    int z = 0;  
    String digits = x + y + "" + z;  
    System.out.println(digits); }
```

В переменную hugeAmount нужно записать число 100500. Используй переменные bigAmount, greatAmount и преобразование строки в число. Для объявления и инициализации hugeAmount используй одну команду.

```
public static void main(String[] args) {  
    String bigAmount = "500";  
    String greatAmount = "100000";  
    int hugeAmount = Integer.parseInt (greatAmount) + Integer.parseInt (bigAmount);  
    System.out.println(hugeAmount); }
```

Метод length()

Метод length() позволяет узнать длину строки – сколько в ней символов или сообщает общее количество символов, содержащихся в строке.

```
String name = "Rome";
```

```
int count = name.length(); // count содержит значение 4
```

Например `int myLength = "Hello World".length();` В этом примере значение `myLength` будет равно 11, так как каждая из строк «Hello» и «World» состоит из 5 символов. Если же добавить пробел, разделяющий два слова, мы получим длину 11.

Методы можно вызывать у всего, что имеет тип `String`, даже у выражения:
`(name + 12).length()`

В методе `main` на экран выводятся значения трех строк.

Внеси изменения в код, чтобы вместо значений строк вывелась длина каждой строки.

```
public static void main(String[] args) {  
    String emptyString = "";  
    int println = emptyString.length();  
  
    System.out.println(emptyString.length());  
    System.out.println("Gomu Gomu no Bazooka!".length());  
    System.out.println((emptyString + 2 + 2 + "22").length()); }  
}
```

Метод `toLowerCase()`

Метод `toLowerCase()` позволяет преобразовать все символы строки в маленькие (строчные)

```
String name = "Rom";  
String name2 = name.toLowerCase(); //name2 содержит строку rom
```

Метод `main` выводит на экран четыре строки. Все они - яркий пример злоупотребления заглавными буквами. Внести изменения в код, чтобы все буквы в этих строках стали строчными

```
public static void main(String[] args) {  
    String title = "Senior Lead Principal Software Engineer Data Architect";  
    String degree = "In college, I Majored in Political Science and Minored in Religious  
Studies.";  
    String career = "Experienced Team Leader with strong Organizational Skills and a  
Successful career in Management.";  
    System.out.println("RESUME".toLowerCase());  
    System.out.println(("TITLE: " + title).toLowerCase());  
    System.out.println(("DEGREE: " + degree).toLowerCase());  
    System.out.println(("CAREER: " + career).toLowerCase()); }  
}
```

Метод `toUpperCase()`

Метод `toUpperCase()` позволяет преобразовать все символы строки в большие (заглавные):

```
String name = "Rom";  
String name2 = name.toUpperCase(); //name2 содержит строку ROM
```

В методе `main` на экран выводятся три строки. Внести изменения в код, чтобы все буквы в этих строках стали заглавными.

```
public static void main(String[] args) {  
    String caps = "if I type in caps ";  
    String usa = "сша";  
    System.out.println(usa.toUpperCase());  
}
```



```
System.out.println("Винни Пух".toUpperCase());  
System.out.println((caps + "they know I mean business").toUpperCase()); }
```

Метод charAt()

Возвращает один символ, находящийся в заданной позиции. Полученный символ может быть присвоен переменной типа char. Например, команда `char myChar = "Hello World".charAt(1);` извлекает символ с индексом 1 и присваивает его myChar. После этого значение переменной myChar будет равно 'e'.

Метод equals()

Используется для проверки равенства двух строк. Он возвращает true, если строки равны, и false в противном случае.

После выполнения следующих команд:

```
boolean equalsOrNot = "This is Jamie".equals("This is Jamie");
```

```
boolean equalsOrNot2 = "This is Jamie".equals("Hello World");
```

переменная equalsOrNot будет равна true, тогда как переменная equalsOrNot2 будет равна false.

Метод split()

Метод разбивает строку на подстроки по разделителям, определяемым пользователем. После разбиения строки метод split() возвращает массив полученных подстрок. Допустим, строку «Peter, John, Andy, David» требуется разбить на подстроки. Это можно сделать так:

```
String names = "Peter, John, Andy, David";
```

```
String[] splitNames = names.split(", ");
```

Сначала значение строки, которую требуется разбить, присваивается переменной names. Затем переменная names используется для вызова метода split(). Метод split() получает один аргумент — разделитель, который будет использоваться для разбиения строки. В нашем примере разделителем является запятая, за которой следует пробел. В результате выполнения этого кода будет получен следующий массив:

```
{"Peter", "John", "Andy", "David"}
```

D. Типы данных (04.07.2023)

В программировании очень часто используется память. И когда говорят о памяти имеют в виду оперативную память компа. Именно в нее загружаются программы для выполнения. Именно из нее берутся исходные данные для выполнения программы. Именно в память записываются результаты выполнения программы. Простейшим способом для программиста использовать память являются переменные.

Переменная (variable) - это именованная область памяти, содержимое которой может изменяться в процессе выполнения программы. Прежде чем использовать переменную ее нужно объявить (создать). Для объявления переменной необходимо указать ее тип и имя следующим образом.

```
type name; // синтаксис объявления переменной.
```

type - это тип переменной.

Он определяет три вещи:

- 1) сколько памяти переменная будет занимать
- 2) какие значения она сможет принимать
- 3) какие операции над ней можно выполнять.

Все типы данных в языке java можно разделить на три категории: логические, символьные и числовые. Эти типы еще называют примитивными. У каждого примитивного типа есть класс обертки, которые хранят основные свойства этого типа.

I категория это логические типы: **bool (boolean)**. Класс обертка также называется Boolean, занимает 1 байт памяти и принимает 1 из 2 значений. true (1) & false (0). Переменную типа bool всегда можно интерпретировать как вопрос, на который всегда можно ответить однозначно да или нет. Переменная и выражения типа bool используются для написания условий управляющих структур (if, else...)

```
// Демонстрация использования значений типа boolean,
class BoolDemo {
public static void main(String[] args) {
boolean b;
b = false;
System.out.println("b равно " + b);
b = true;
System.out.println("b равно " + b);
// Значение boolean может управлять оператором if.
if(b) System.out.println("Данная строка кода выполняется.");
b = false;
if(b) System.out.println("Данная строка кода не выполняется.");
// Результатом операции отношения является значение boolean.
System.out.println("10 > 9 равно " + (10 > 9)); }}
```

В данном проекте создается программа, которая вычисляет расстояние (в метрах) между наблюдателем и местом вспышки молнии. Звук распространяется по воздуху со скоростью примерно 335 метров в секунду. Таким образом, зная интервал между моментом, когда появилась вспышка молнии, и моментом, когда был услышан звук, можно рассчитать расстояние до места вспышки. Предположим, что временной интервал составляет 7,2 секунды.

```
/*
Упражнение 2.1. Расчет расстояния до места вспышки молнии, звук которого был
услышан через 7.2 секунды .
*/
class Sound {
public static void main(String[] args) {
double dist;
dist = 7.2 * 335;
System.out.println("Место вспышки молнии находится на расстоянии "
+ dist + " метров.");}}
```

II категория - символьные типы также включает в себя всего один тип данных. Это **char (character)** - символ. Занимает два байта памяти. И содержит код символов в кодировке Unicode. Всего возможных символом 65536. Т.е. переменная типа char содержит 1 из 65536 символов. "Hello" - строковый тип

```
// С символьными переменными можно обращаться как с целочисленными.
class CharArithDemo {
public static void main(String[] args) {
char ch;
ch = 'X';
System.out.println("ch содержит " + ch);
ch++; // инкрементирование ch можно инкрементировать
System.out.println("ch теперь содержит " + ch);
ch = 90; // присваивание ch значения Z. Переменной типа char можно присвоить
целочисленное значение
```

```
System.out.println("ch теперь содержит " + ch);}}
```

III категория - числовые типы. Делятся на целочисленные и вещественные. Целочисленные предназначены для хранения целых чисел. А вещественные для хранения дробных чисел или чисел с плавающей запятой. В свою очередь целочисленные типы делятся на **беззнаковые (unsigned)** - предназначенные для хранения только положительных целых чисел и **знаковые (signed)** - могут хранить как положительные так и отрицательные целые числа

К **целочисленным типам (Integer types)** относятся:

byte - занимает 1 байт памяти и принимает значения в диапазоне от -128 до 127. $2^8 \dots 2^8 - 1$

1 byte = 8 bit; bit - binary digit (двоичная цифра) 0 или 1. В беззнаковой переменной все биты являются значащими т.е. содержат модуль числа. В знаковой же переменной старший бит является знаковым т.е. содержит знак числа и для хранения модуля числа остается на 1 бит меньше. Sign: 0 '+' & 1 '-'.

short - занимает 2 байта (16 bit) и принимает значения в диапазоне -32 768 до 32 767. $2^{15} \dots 2^{15} - 1$

int (integer) - целое число. Занимает 4 байта (32 bit) в диапазоне -2 147 483 648 до 2 147 483 647. $2^{31} \dots 2^{31} - 1$

long - 8 байта (64 bit) - самый целочисленный примитивный тип данных - $2^{63} \dots 2^{63} - 1$. $9,223372 \times 10^{18}$ до $9,223371 \times 10^{18}$

$10 \times 10 \times 10 = 10^3 = 1000$

000
001
010
011
100
101
110
111

чтобы посчитать кол-во комбинаций из 3 битов $2 * 2 * 2 = 2^3$:

$$N = a^n$$

N - кол-во комбинаций, a - основание системы счисления, n - разрядность числа

Д/з Используя классы обертки и метод format вывести на экран свойства числовых данных в следующем виде: "Переменная типа **byte** занимает 1 Байт памяти (8 Бит памяти), и принимает значение в диапазоне от -128 до 127"

```
public class Main {  
    public static void main(String[] args) {  
        System.out.format("Переменная типа " + Byte.TYPE + " занимает " + Byte.BYTES  
+ " Байт памяти " + "(" + Byte.SIZE + " Бит памяти), и принимает значение в  
диапазоне от " + Byte.MIN_VALUE + " до " + Byte.MAX_VALUE + ".");  
    }  
}
```

Верное решение: `.formatted()` - форматирование строки. Нужно поставить метки экранированных процессов. `%s` - текстовое значение, `%d` числовое значение.

```
System.out.println("Переменная типа %s занимает %d Байт памяти (%d Бит памяти), и принимает значение в диапазоне от %d до %d".formatted( "int", Integer.BYTES,Integer.SIZE, Integer.MIN_VALUE, Integer.MAX_VALUE ));
System.out.println("Переменная типа %s занимает %d Байт памяти (%d Бит памяти), и принимает значение в диапазоне от %d до %d".formatted( "byte", Byte.BYTES, Byte.SIZE, Byte.MIN_VALUE, Byte.MAX_VALUE ));
System.out.println("Переменная типа %s занимает %d Байт памяти (%d Бит памяти), и принимает значение в диапазоне от %d до %d".formatted( "short", Short.BYTES, Short.SIZE, Short.MIN_VALUE, Short.MAX_VALUE ));
System.out.println("Переменная типа %s занимает %d Байт памяти (%d Бит памяти), и принимает значение в диапазоне от %d до %d".formatted( "long", Long.BYTES, Long.SIZE, Long.MIN_VALUE, Long.MAX_VALUE ));
```

или

```
System.out.format("Переменная типа %s занимает %d Байт памяти (%d Бит памяти), и принимает значение в диапазоне от %d до %d", "int", Integer.BYTES,Integer.SIZE, Integer.MIN_VALUE, Integer.MAX_VALUE );
```

GK__Java_urok_03

Операнд – может быть литералом, переменной или выражением, над которыми производится операция.

Операции могут быть объединены в выражения.

Есть несколько операций, перегруженных по умолчанию:

- `+` – используется для сложения чисел и конкатенации строк;
- `&` – используется для побитовой операции с числами и для логического И;
- `|` – используется для побитовой операции с числами и для логического ИЛИ;
- `^` – используется для побитовой операции с числами и для логического исключающего ИЛИ;
- `==` – сравнивает любые типы;
- `!=` – сравнивает любые типы

Арифметические операции

Обозначение	Описание	Пример
+	Унарный плюс, не меняет значение операнда.	<pre>int i = -1; i = +i; System.out.println(i);</pre>
	Бинарный плюс, суммирует операнды.	<pre>int i = -1; i = i + 3; System.out.println(i);</pre>
-	Унарный минус – меняет знак операнда на противоположный.	<pre>int i = 1; i = -i; System.out.println(i);</pre>
	Бинарный минус – вычитает правый операнд из левого.	<pre>int i = 3; i = i - 2; System.out.println(i);</pre>
*	Умножение операндов	<pre>int i = 2; i = i * 2; System.out.println(i);</pre>
/	Деление левого операнда на правый.	<pre>int i = 4; i = i / 2; System.out.println(i);</pre>
%	Вычисление остатка от деления левого операнда на правый.	<pre>int i = 3; i = i % 2; System.out.println(i);</pre>
++	Бинарный оператор увеличения значения переменной на 1. Инкремент.	i++; заменяет выражение i = i + 1;
--	Бинарный оператор уменьшения значения переменной на 1. Декремент.	i--; заменяет выражение i = i - 1;

Побитовые операции

ВАЖНО!!! В качестве операндов в побитовых операциях могут быть использованы только целые примитивные типы или их классы-обертки.

Операция	Описание	Пример	Результат
&	И (AND)	<p>В каждом разряде результата получаем значащий бит, в случае если присутствует значащий бит в левом и правом операнде.</p> <pre>int i = 123 & 456;</pre> <p>Битовое представление</p> <pre>000000000000000000000000001111011 & 0000000000000000000000000111001000 = 000000000000000000000000001001000</pre>	72
	ИЛИ (OR)	<p>В каждом разряде результата получаем значащий бит, в случае если присутствует значащий бит в левом или правом операнде.</p> <pre>final int one = 123; final int two = 456; int i = one two;</pre> <p>Битовое представление</p> <pre>000000000000000000000000001111011 00000000000000000000000000111001000 = 000000000000000000000000011111011</pre>	507

Операция	Описание	Пример	Результат
<code>^</code>	исключающее ИЛИ (XOR)	В каждом разряде результата получаем значащий бит, в случае если присутствует значащий бит в левом или правом операнде. Если бит есть в разряде обоих операндов, в результате получаем 0. <code>int i = 123 ^ 456;</code> Битовое представление 000000000000000000000000000000001111011 ^ 00000000000000000000000000000000111001000 = 00000000000000000000000000000000110110011	435
<code>~</code>	побитовое отрицание (NOT)	Инвертирует все биты числа на противоположные (включая знаковый бит) <code>int i = ~123;</code> Битовое представление: ~ 000000000000000000000000000000001111011 = 111111111111111111111111111111110000100	-124
<code><<</code>	сдвиг влево	Смещает биты левого операнда влево на количество бит указанном в правом операнде, заполняя справа нулем.	
<code>>></code>	сдвиг вправо	Смещает биты левого операнда вправо на количество бит указанном в правом операнде, заполняя разряды справа нулем.	
<code>>>></code>	без знаковый сдвиг вправо	Смещает биты левого операнда вправо на количество бит указанном в правом операнде, заполняя слева нулем игнорируя битовый сдвиг.	

Операции сравнения

Результат операций сравнения всегда имеет тип **boolean**

Обозначение	Условие	Пример
<code>></code>	операнд слева больше, чем операнд справа	<code>int a = 7;</code> <code>boolean r = a > 5;</code>
<code><</code>	операнд слева меньше, чем операнд справа	<code>int a = 7;</code> <code>boolean r = a < 5;</code>
<code>>=</code>	операнд слева больше или равен операнду справа	<code>int a = 7;</code> <code>boolean r = a >= 5;</code>
<code><=</code>	операнд слева меньше или равен операнду справа	<code>int a = 7;</code> <code>boolean r = a <= 5;</code>
<code>==</code>	операнд слева в точности равен операнду справа	<code>int a = 7;</code> <code>boolean r = a == 5;</code>
<code>!=</code>	операнд слева не равен операнду справа	<code>int a = 7;</code> <code>boolean r = a != 5;</code>

Важный момент 1: Операторы, состоящие из двух символов, разрывать нельзя.

Важный момент 2: Обратите внимание, что операторов `=>` и `=<` нет: есть только `<=` и `>=`. Если вы напишете `a =< 3`, ваш код просто не скомпилируется.

Важный момент 3: В Java вы не можете записать выражение вида `18 < age < 65`. Ведь у выражения `18 < age` будет значение `true` или `false`. А сравнивать `true < 65` (разные типы) нельзя. По крайней мере, в языке Java.

Ввести с клавиатуры три целых числа. Определить, есть ли среди них хотя бы одна пара равных между собой чисел. Если такая пара существует, вывести на экран числа через пробел. Если все три числа равны между собой, то вывести все три.

```
int age = 70;
boolean isSenior = (age > 65);
if (isSenior)
    System.out.println("Пора на пенсию");
```

```
public class Solution {
    private static boolean a;
    private static boolean b;
    private static boolean c;
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int a = scanner.nextInt();
        int b = scanner.nextInt();
        int c = scanner.nextInt();
        if (a==b && b==c && c==a) {
            System.out.println(a + " " + b + " " + c);
        } else if (a==c) {
            System.out.println(a + " " + c);
        } else if (a==b) {
            System.out.println(a + " " + b);
        } else if (b==c) {
            System.out.println(b + " " + c);
        } else {}
    }
}
```

Логические операции

Логические операции применяются только к операндам **типа boolean**. Результат логических операций имеет **тип boolean**.

Обозначение операции	Описание
&	Логическое И
	Логическое ИЛИ
^	Логическое исключающее ИЛИ
	Оператор быстрой оценки ИЛИ*
&&	Оператор быстрой оценки И*
!	Отрицание

* Операторы быстрой оценки не вычисляют результат выражения в правом операнде, если результат операции можно определить по значению первого операнда.

Программа считывает с клавиатуры значение температуры тела и выдает сообщение о том, что температура тела высокая, низкая или нормальная, в зависимости от условий. В классе объявлены две булевы переменные isHigh (высокая температура) и isLow (низкая), в которые нужно вынести соответствующие условия и вместо выражений сравнения использовать эти переменные.

```
public class Solution {
    private static boolean isHigh;
    private static boolean isLow;
```



```

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    double bodyTemperature = scanner.nextDouble();

    isHigh = (bodyTemperature > 37);

    isLow = (bodyTemperature < 36);

    if (isHigh) {

        System.out.println("температура тела высокая");
    }

    else if (isLow){

        System.out.println("температура тела низкая");
    }

    else {

        System.out.println("температура тела нормальная");
    }
}
}

```

Напишем программу, которая будет просчитывать возможность существования треугольника на основе длин его сторон.

```

public class Solution {
    private static boolean a;
    private static boolean b;
    private static boolean c;
    private static final String TRIANGLE_EXISTS = "треугольник существует";
    private static final String TRIANGLE_NOT_EXISTS = "треугольник не существует";
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int a = scanner.nextInt();
        int b = scanner.nextInt();
        int c = scanner.nextInt();
        if ( a<(b+c) && b<(c+a) && c<(a+b) ) {
            System.out.println(TRIANGLE_EXISTS);
        } else if (a >= (b+c) || b>= (c+a) || c >= (a+b)) {
            System.out.println(TRIANGLE_NOT_EXISTS);
        }
    }
}

```

Напишем программу, которая будет считывать с клавиатуры возраст. Если возраст от 20 до 60 (включительно), то выводить ничего не нужно, иначе - вывести фразу "можно не работать". Сделать это можно (и нужно!) с помощью только одного оператора if без else. Подсказка: используй логический оператор "||" (или).

```

public class Solution {
    private static boolean age;
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int age = scanner.nextInt();
    }
}

```

```

    if ((age < 20 || age > 60) && (age >= 20 || age <= 60) ) {
        System.out.println("можно не работать");
    }
}
}

```

Пример для И: `boolean t = 5 < 3 && 5 > 3`

В данном примере в результате вычисления выражения левого операнда получим `false`, поэтому для вычисления результата операции нет необходимости вычислять правый операнд.

Пример для ИЛИ: `boolean t = (2 == 2) || 3 != 2`

В данном примере в результате вычисления выражения левого операнда получим `true`, поэтому для вычисления результата операции ИЛИ нет необходимости вычислять правый операнд.

Таблица истинности логических операций

Операнд 1	Операнд 2	Операция			
			&	^	!Операнд1
false	false	false	false	false	true
true	false	true	false	true	false
false	true	true	false	true	true
true	true	true	true	false	false

Ввести с клавиатуры два целых числа, которые будут координатами точки (первое считанное число - это координата "x", а второе - координата "y"). Известно, что точка не лежит на координатных осях OX и OY. Вывести на экран номер координатной четверти, в которой находится данная точка.

```

public class Solution {
    private static boolean x;
    private static boolean y;
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int x = scanner.nextInt();
        int y = scanner.nextInt();
        if (x>0 && y>0) {
            System.out.println("1");
        } else if (x<0 && y>0) {
            System.out.println("2");
        } else if (x<0 && y<0) {
            System.out.println("3");
        } else if (x>0 && y<0) {
            System.out.println("4");
        }
    }
}

```

Операция присваивания

Для сохранения в переменной значения литерала, переменной или значения выражения используется **операция присваивания**.

Синтаксис: переменная = выражение;

Тернарный оператор

Тернарный оператор содержит три операнда. Результатом операции может быть второй или третий операнд. Первый операнд должен иметь тип boolean.

Синтаксис: выражение1 ? выражение2 : выражение3;

Если выражение1 истинно, то выполняется выражение2 иначе выполняется выражение3. После условия следует знак вопроса, а два выражения разделены двоеточием. Основное отличие тернарного оператора от оператора if-else в том, что тернарный оператор является выражением, а значит его результат можно чему-нибудь присвоить.

Например, мы хотим вычислить минимум двух чисел. С использованием тернарного оператора этот код будет выглядеть так:

```
int a = 2;
int b = 3;
int min = a < b ? a : b;
```

Или, допустим, вам нужно присвоить переменной разные значения в зависимости от какого-то условия. Как это сделать?

Вариант первый — воспользоваться оператором if-else:

```
int age = 25;
int money;
if (age > 30)
    money = 100;
else
    money = 50;
```

Второй вариант — использовать тернарный оператор, то есть сокращенную запись оператора if-else:

```
int age = 25;
int money = age > 30 ? 100 : 50;
```

У нас есть программа, которая считывает с клавиатуры два числа и выводит на экран большее из них. Если числа одинаковые, то выводится любое.

Перепиши программу с использованием тернарного оператора, чтобы ее функционал остался без изменений.

```
public class Solution {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int numberA = scanner.nextInt();
        int numberB = scanner.nextInt();
        int max = numberA > numberB ? numberA : numberB;
        System.out.println (max);
    }
}
```

Эта программа написана с использованием тернарного оператора, но не совсем понятно, что она делает. Думаем, если избавиться от тернарного оператора, то станет намного понятнее. Перепиши программу без использования тернарного оператора.

```
public class Solution {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int number = scanner.nextInt();
        if (number < 5)
            System.out.println("число меньше 5");
        else if (number > 5)
            System.out.println("число больше 5");
        else
            System.out.println("число равно 5");
    }
}
```

Conditional Ternary (тернарный оператор)

Синтаксис тернарного оператора: Условие - это операция сравнения.

```
condition ? value 1 : value 2;
```

Тернарный оператор - возвращает одно из двух значений в зависимости от условия, если условие вернуло true возвращается value_1, в противном случае value_2. value_1 и value_2 обязательно должны быть одного типа, в противном случае поведение не определено. Как раз **возвращаемое значение делает тернарное значение тернарным** и это позволяет сделать его частью выражения.

```

*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****

      *****      *****
      *****      *****
      *****      *****
      *****      *****
      *****      *****
      *****      *****

*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****

      *****      *****
      *****      *****
      *****      *****
      *****      *****
      *****      *****
      *****      *****

*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****

```

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.println("Размер фигуры: ");
        int n = kb.nextInt();

        for (int i = 0; i < n; i++)
        {
            for (int j=0; j<n; j++) {
                for (int k=0; k<n;k++) {
                    for (int l=0; l<n; l++) {
                        System.out.print((i%2==k%2) ? "*" : " ");
                    }
                }
            }
        }
    }
}
```

```

    }
    System.out.println();
  }
}
}
}

```

Сравнение вещественных чисел

Если два вещественных числа отличаются на очень маленькое значение, значит можно считать их равными. Пример:

```

double a = 1.000001;
double b = 1.000002;
if ( (b - a) < 0.0001 )
    System.out.println("Числа равны");
else
    System.out.println("Числа не равны");

```

В Java есть специальный метод для вычисления модуля числа — `Math.abs()`:
`int m = Math.abs(значение);`

```

double a = 1.000001;
double b = 1.000002;
if ( Math.abs(b - a) < 0.0001 )
    System.out.println("Числа равны");
else
    System.out.println("Числа не равны");

```

Напиши программу, которая считывает с клавиатуры два вещественных числа (double) и выдает сообщение о том, равны ли эти числа с точностью до одной миллионной.

```

public class Solution {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        double a = scanner.nextDouble();
        double b = scanner.nextDouble();
        if (Math.abs(b - a) < 0.000001)
            System.out.println("числа равны");
        else
            System.out.println("числа не равны");
    }
}

```

Приоритеты операций

Операции	Описание
() []	Круглые и квадратный скобки
++ -- + - ~ !	Декремент, инкремент, унарный плюс, унарный минус, поразрядное отрицание, логическое отрицание

Операции	Описание
* / %	Умножение, деление, остаток от деления
-	Сложение вычитание
>> >>> <<	Побитовые сдвиги: вправо, беззнаковый вправо, побитовый сдвиг влево
> >= < <=	Сравнение на: больше, больше или равно, меньше, меньше или равно
== !=	Сравнение на равенство, сравнение на неравенство
& ^ &&	Поразрядное и логическое: И, исключающее ИЛИ, ИЛИ, быстрой оценки И, ИЛИ
? :	Тернарная условная операция
операция=	Комбинированные операторы

Управляющие конструкции

Для реализации алгоритма ветвления в коде в Java присутствует два условных оператора, это оператор **if ... else** и оператор **switch**. Для реализации циклических алгоритмов есть три оператора, это **for**, **while**, **do..while**.

Синтаксис: if (выражение) операция;

В круглых скобках может быть указано любое выражение, результатом которого должен быть тип `boolean`. В случае если в скобках выражение примет значение `true`, то будет выполнена «операция» следующая за круглыми скобками, иначе управление программы перейдет к следующей строке кода

Например `if(10 < 11) System.out.println("10 меньше 11");`

```

/*
Демонстрация использования оператора if.
Назовите этот файл IfDemo.java.
*/
class IfDemo {
public static void main(String[] args) {
int a, b, c;
a = 2;
b = 3;
if(a < b) System.out.println("Значение a меньше значения b");
// Следующий оператор ничего не отобразит.
if(a == b) System.out.println("Это вы не увидите");
System.out.println();
c = a - b; // Переменная c содержит -1
System.out.println("Переменная c содержит -1");
if(c >= 0) System.out.println("Значение c неотрицательное");
if(c < 0) System.out.println("Значение c отрицательное");
System.out.println();
c = b - a; // Теперь переменная c содержит 1
System.out.println("Переменная c содержит 1");
if(c >= 0) System.out.println("Значение c неотрицательное");
if(c < 0) System.out.println("Значение c отрицательное");
}
}

```

Совместно с оператором if возможно использоваться ключевого слова else.

Синтаксис: if (выражение) операция1;
else операция2;

В случае если в скобках «выражение» примет значение false, то будет выполнена «операция 2», «операция 1» выполнена не будет.

Оператор if может использоваться совместно с else, для создания логических цепочек. Программа предлагает игроку угадать букву, после чего читает символ с клавиатуры. Затем с помощью оператора if прочитанный символ сравнивается с ответом answer — буквой K в данном случае. Если была введена буква K, тогда отображается сообщение. При опробовании программы не забывайте, что буква K должна вводиться в верхнем регистре.

```
// Игра в угадывание буквы .
class Guess {
public static void main(String[] args)
throws java.io.IOException {
char ch, answer 'K';
System.out.println("Задумана буква между A и Z.");
System.out.print("Попробуйте ее угадать: ");
ch = (char) System.in.read();
if(ch == answer) System.out.println(
// чтение символа с клавиатуры
и** Правильно ** и
}
}
```

```
// Игра в угадывание буквы , версия 2.
class Guess2 {
public static void main(String[] args)
throws java.io.IOException {
char ch, answer = 'K';
System.out.println("Задумана буква между A и Z.");
System.out.print("Попробуйте ее угадать: ");
ch = (char) System.in.read(); // чтение символа с клавиатуры
if(ch == answer) System.out.println("*** Правильно ***");
else System.out.println("...Увы , не угадали.");
}}
```

С помощью вложенного оператора if можно улучшить игру в угадывание буквы. Следующее дополнение программы выдает игроку подсказку о том, где находится правильная буква, в случае ввода некорректного предположения.

```
// Игра в угадывание буквы , версия 3.
class Guess3 {
public static void main(String[] args)
throws java.io.IOException {
char ch, answer = 'K';
System.out.println("Задумана буква между A и Z.");
System.out.print("Попробуйте ее угадать: ");
ch = (char) System.in.read(); // чтение символа с клавиатуры
if(ch == answer) System.out.println(** Правильно **);
```



```

else {
    System.out.print .Увы , не угадали. Задуманная буква находится ");
    if( ch < answer) System.out.println("дальше по алфавиту.");
    else System.out.println("ближе по алфавиту.");
}
}
}

```

Е. Вещественные типы (06.07.2023)

В языке java сущ всего два вещественных типа: float & double.

Float - вещественный тип одинарной точности. Занимает 4 байта памяти. В диапазоне от $1.401298e-45$ до $3.402823e+38$. Переменная типа float содержит 45 знаков после запятой, с точностью до 7 знаков.

e (exponent) - основание системы счисления. Пределы можно записать как $1.401298 \cdot 10^{-45}$ до $3.402823 \cdot 10^{38}$.
 $0,0038 = 38/10000 = 38/10^4 = 38 \cdot 10^{-4} = 38 e^{-4}$

S	Exponent	Mantissa
---	----------	----------

S (sign) - знак числа.

Exponent - плавающая экспонента, определяет на сколько знаков и в какую сторону смещена запятая.

Mantissa - это дробная часть дробного числа, целая часть всегда равна 1.

Double - вещественный тип двойной точности. Занимает 8 байт памяти. В диапазоне от $4.900000E - 324$ до $1.797693E + 308$. Переменная типа double содержит (хранит) 324 знака после запятой, отображает с точностью до 15 знаков.

Все вещественные типы являются только знаком. Не существует беззнаковых вещественных типов. Знак является неотъемлемой частью дробного числа.

Операции над целыми числами выполняет блок процессора **ALU (Arithmetic-Logic Unit)** арифметико логическое устройство. Операции над **FPU (Floating-Point Unit)** блок операции с плавающей точкой.

Вещественные типы могут хранить оч большие и оч маленькие числа. Но эти числа как правило являются не точными.

Целая и дробная часть числа всегда разделяются . (точкой)

```

/*
Использование теоремы Пифагора для нахождения
длины гипотенузы по длинам двух катетов.
*/
class Hypot {
    public static void main(String[] args) {
        double x, y, z;
        x = 3;
        y = 4;
        z = Math.sqrt(x*x + y*y); //Обратите внимание на вызов метода sqrt(). Он предварен
именем класса, членом которого он является.
        System.out.println("Длина гипотенузы : " +z);
    }
}

```

Ф. Преобразование типов (11.07.2023)

Существует явное и неявное преобразование типов.

Явное преобразование выполняет программист, а неявное компилятор. Для того чтобы преобразовать значение в другой тип данных необходимо желаемый тип данных

написать в круглых скобках перед значением. При этом значение так же иногда лучше взять в круглые скобки.

int grn = (int) money; - money тут переменная, чтобы сохранить это значение, обращаться к ней по имени.

Как явные так и неявные преобразования бывают от меньшего к большему, так и от большего к меньшему. При чем второе может привести к потере данных. Поэтому компилятор запрещает все неявные преобразования от большего к меньшему.

Неявные преобразования выполняет компилятор, а точнее операторы. Каждый оператор пытается привести свои операнды к наибольшему типу данных. Для того чтобы вернуть максимально точный результат.

Имя переменной нужно для того чтобы к ней можно было обращаться. Для именования переменных используются идентификаторы составленные по определенным правилам:

1. имя переменной может состоять из символов латинского алфавита, символов цифр и символа подчеркивания. ABC...Zabc....z012...9_;
2. имя переменной никогда не может начинаться с символом цифры
3. имена переменных регистрозависимы, т.е. большие и маленькие буквы отличаются
4. для именования переменных нельзя использовать ключевые слова языка java. например int. но если написать с большой буквы, то будет все в порядке Int.

Важное правило! Имена переменных обязательно должны быть осмыслены. Т.е. по имени переменной должно становиться понятно что в ней храниться. Это делает код логичным и понятным.

Д/з 1) Преобразование числа в денежный формат из гривны в рубли

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        System.out.println("Преобразование числа в денежный формат");
        //создаем сканер при помощи которого будем вводить данные с клавиатуры
        Scanner kb = new Scanner(System.in);
        //объявляем переменную в которую будет вводиться денежная сумма
        double money;
        //перед вводом пол-ль обязат должен увидеть приглашение на ввод
        System.out.print("Введите дробное число: ");
        //обеспечиваем ввод значения пользователя с клавиатуры
        money = kb.nextDouble();
        money+=1e-5; //прибавляем 1*10^-5 чтобы скорректировать число в периоде
        //проверяем ввод
        System.out.println(money);
        //разделение числа
        int grn = (int) money; //явно преобразуем переменную money в тип int
        int cop = (int)((money-grn)*100);
        System.out.printf(money + " - это %d грн. %d коп.\n", grn, cop);
    }
}
```

2) сделать конвертер валют рубли в доллары, доллары в евро. **Инициализация (int)**

```
import java.util.Scanner;
import java.text.DecimalFormat;
```

```

public class Main {
    public static void main(String[] args) {

        double rub, dollar, euro;
        DecimalFormat r = new DecimalFormat("##.##");
        DecimalFormat d = new DecimalFormat("##.##");
        DecimalFormat e = new DecimalFormat("##.##");
        Scanner kb = new Scanner(System.in);
        System.out.println("Конвертер валют");
        System.out.print("Введите число (руб.): ");
        rub = kb.nextDouble();

        dollar = rub / 90.63;
        System.out.println(rub + " ₺ = " + d.format(dollar) + " $");
        euro = dollar * 0.90726;
        System.out.println(d.format(dollar) + " $ = " + e.format(euro) + " €");
        rub = euro * 99.89;
        System.out.println(e.format(euro) + " € = " + r.format(rub) + " ₺");
    }
}

```

При объявлении переменной под нее просто выделяется память. Возможно раннее в этой памяти что-то хранилось какое-то неизвестное значение. Такие неизвестные значения называют **мусором**. Для того чтобы убрать из переменной мусор, ее нужно проинициализировать. **Инициализация** - присвоение начального значения. Инициализацию как правило выполняют при объявлении переменной. Но ее также можно выполнить после объявления переменной. Инициализация это когда в переменную в первый раз что-то записывается. От слова Init - начало. В языке Java абсолютно все должно быть проинициализировано. В противном случае возникают ошибки на этапе компиляции. Абсолютно каждой переменной должно быть значение, если не задали, значит там мусор, а мусор компилятор не пропускает.

Ошибки связанные с переменной.

1. **java: cannot find symbol** - возникает если используем не объявленную переменную.

```

int a=2;
System.out.println(b); double b;

```

2. **Variable is already defined in the scope** - возникает при повторном объявлении такой же переменной.

```

int a = 2;
int b = 3;
System.out.println(b);
double b;

```

Константы

Константа - это именованная область памяти, содержащая которой не может изменяться в процессе выполнения программы. Константу очень легко сделать из переменной, написав перед ее объявлением ключевое слово **final**.

```

int speed = 0;
final int max_speed = 250; //максимальную скорость мы не можем изменить - она постоянна. Если написали final, то переменную мы обязаны писать константу в верхнем регистре MAX_SPEED и будет значение final int MAX_SPEED = 250;

```

speed = 60; //мы можем изменять переменную значения, потому что она меняется

Кроме именованных констант существуют также числовые, символьные и строковые константы.

Числовая константа - это просто число в исходном коде программы.

Например 5; - это числовая константа типа int. Любой тип данных является объектом например объявить System.out.println(((Object)5).getClass().getSimpleName());

Тут 5 - это значение.

Object - абсолютное любое значение на языке java является объектом и его всегда можно преобразовать в объект.

getClass() - в классе Object есть метод getClass который позволяет определить конкретный тип значения, значение может быть буква или число

getSimpleName - Этот метод (метод-это функция) возвращает строку содержащую имя типа (название типа данных).

Значение (value) - любое значение может быть переменное (variable) или постоянное (const) или (final) Абсолютно у каждого значения есть тип или Абсолютно любое значение какого то типа.

System.out.println(((Object)5l).getClass().getSimpleName()); 5l или L из константы типа int делаем константу типа Long

System.out.println(((Object)5.).getClass().getSimpleName()); будет тип Double мы указали целую часть не дробную

System.out.println(((Object).5).getClass().getSimpleName());

System.out.println(.623); тут не указали целую часть десятичной дроби, то будет 0.623

В short преобразовать нельзя, можно только явным преобразованием в short.

Символьные константы - это один единственный символ заключенный в одинарные кавычки. Символьные константы типа **char**. Занимает 2 байта. Символьные константы можно использовать для сравнения с символьными переменными. System.out.println(((Object)'').getClass().getSimpleName()); Character

Строковая константа - это последовательность символов заключенная в двойные кавычки. System.out.println(((Object)"".getClass().getSimpleName()); String

Числовые, символьные и строковые константы еще называют литералами. **Литералы** - это значение которое воспринимается как **есть**. Литералы Java могут относиться к любому примитивному типу данных. Способ представления каждого литерала зависит от его типа. Как объяснялось ранее, символьные константы заключаются в одинарные кавычки, например, 'a' и ' % '. Целочисленные литералы указываются как числа без дробных частей подобно 10 и -100. Литералы с плавающей точкой требуют применения десятичной точки, за ко плавающей точкой также разрешено использовать экспоненциальную запись. Целочисленные литералы по умолчанию имеют тип int. Чтобы указать литерал типа long, понадобится добавить к константе букву l или L. Например, 12 — литерал типа int, а 12L - литерал типа long. По умолчанию литералы с плавающей точкой имеют тип double. Чтобы указать литерал типа float, необходимо добавить к константе букву F или f. Например, 10.19F - литерал типа float.

Тип double

Тип double используется для хранения вещественных чисел.

`double имя;`

```
int t = 1000;
double x = t * t; // В переменной x хранится значение 1000000.0
```

Если в каком-то выражении участвуют целое и вещественное число, целое сначала преобразуется в вещественное и только потом взаимодействует с другим вещественным числом.

```
int t = 1000;
double x = t * 5.0; //В переменной x хранится значение 5000.0
```

Также компилятор требует, чтобы этот факт программист задокументировал явно (чтобы другие программисты понимали, что тут происходит отбрасывание дробной части). Общий вид этого выражения в коде такой (превратить double в int):

`целочисленная_переменная = (int)(вещественное_число);`

Также значение double можно преобразовать к типу float.
`float num1 = (float) 20.9;`

```
double a = 5.999;
int b = 2;
int x = (int)(a * b); //В переменной x хранится значение 11
```

Ввести с клавиатуры положительное целое число radius. Это будет радиус окружности. Вывести на экран площадь круга, рассчитанную по формуле: $S = \pi * \text{radius} * \text{radius}$. Результатом должно стать целое число (тип int). Для этого нужно привести к типу int результат умножения (отбросить дробную часть, округлив вниз до целого числа). В качестве значения π используй 3.14.

```
import java.util.Scanner;
/*
Площадь круга
*/
public class Solution {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        int radius = kb.nextInt();
        double pi = 3.14;
        int S = (int)(pi * radius * radius);
        System.out.println(S);
    }
}
```

Деление целых и вещественных чисел в Java `double d = 5.0 / 2;` `double d = 5 / 2.0;`
`double d = 5.0 / 2.0;`

Команда	Порядок выполнения	Результат
---------	--------------------	-----------

<pre>int a = 5; int b = 2; double d = 1.0 * a / b;</pre>	$(1.0 * a) / b;$	2.5
--	------------------	-----

Давай разделим ящик колы на кабинет программистов. Для этого напишем программу, в которой: Нужно ввести с клавиатуры два целых числа. Первое число - количество банок колы в ящике. Второе - количество людей в кабинете. Вывести на экран результат деления первого числа на второе.

```
import java.util.Scanner;
/*
Share a Coke
*/
public class Solution {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        int banka = kb.nextInt();
        int chel = kb.nextInt();
        double d = banka * 1.0 / chel;
        System.out.println(d);
    }
}
```

Метод Math.round()

Метод Math.round() округляет число до ближайшего целого:

```
long x = Math.round(вещественное_число)
```

Но, как говорится, есть нюанс: результат работы этого метода — целочисленный тип long (не int). Поэтому чтобы присвоить результат в переменную типа int, программист должен явно указать компилятору, что он согласен с возможной потерей данных (вдруг число не помещается в тип int).

```
int x = (int) Math.round(вещественное_число)
```

Метод Math.ceil()

Метод Math.ceil() округляет число до целого вверх, примеры:

Команда	Результат
<pre>int x = (int) Math.ceil(4.1);</pre>	5

Метод Math.floor()

Метод Math.floor() округляет число до целого вниз, примеры:

Команда	Результат
<pre>int x = (int) Math.floor(4.1);</pre>	4

Хотя, для округления числа до целого вниз, будет проще использовать просто оператор приведения типа — (int)

Команда	Результат
<code>int x = (int) 4.9</code>	4

Поэтому сделаем конвертер величин скорости из м/с в км/ч. Исходную величину скорости ветра в м/с нужно получить, считав ее как целое число из клавиатуры. Необходимо вывести на экран скорость ветра в км/ч, округленную до ближайшего целого числа. Для округления необходимо использовать метод `Math.round()`.

```
import java.util.Scanner;
/*
Скорость ветра
*/
public class Solution {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        int m = kb.nextInt();
        double om = 3.6;
        double d = 1.0 * om * m;
        int speed = (int) Math.round(d);
        System.out.println(speed);
    }
}
```

В методе `main()` есть переменная `double glass = 0.5`, которая символизирует наполовину заполненный стакан. Для пессимиста он наполовину пуст, а для оптимиста - наполовину полон. Необходимо считать с клавиатуры данные типа `boolean`, используя метод `nextBoolean()` объекта типа `Scanner`. В зависимости от полученных данных, округлить переменную `glass`: до целого числа вниз (0), если пессимист (`false`) и до целого числа вверх (1), если оптимист (`true`). Результат вывести на экран.

```
import java.util.Scanner;

/*
Стакан наполовину пуст или наполовину полон?
*/
public class Solution {
    public static void main(String[] args) {
        double glass = 0.5;
        Scanner kb = new Scanner(System.in);
        boolean gl = kb.nextBoolean();
        double d = 1.0 * glass;
        int s = (int) Math.ceil(d);
        int f = (int) Math.floor(d);

        if (gl==true)
        {
            System.out.println(s);
        }
        else {
            System.out.println(f);
        }
    }
}
```


Интересный факт о strictfp

В Java есть специальное ключевое слово strictfp (strict floating point), которого нет в других языках программирования. Оно ухудшает точность работы с вещественными числами.

G. Операторы (13.07.2023)

Программа на языке java состоит из выражений. **Выражения (expression)** - это синтаксическая конструкция состоящая из операндов и операторов.

Операнды - это объекты над которыми выполняются действия. В качестве операндов обычно выступают переменные и константы, т.е. значения.

Операторы - это объекты (элементы выражения) которые показывают какое именно действие нужно выполнить над операндами. Операторы как правило обозначаются одним или двумя специальными символами.

$a = b + c$. $+$ и $=$ это оператор, a , b , c , - операнд



Выражения как правило заканчиваются точкой с запятой. Операторы бывают унарные, бинарные и тернарные

Унарные (Unary) - выполняют действия над операндом -21 ; $33-21$; $7*8$; $*8$;

Бинарные (Binary) - могут работать только с двумя операндами. У бинарных операторов всегда есть два операнда левый и правый.

Тернарные (Ternary) - работает только с тремя операндами.

В языке java есть всего один тернарный оператор

Все операторы можно поделить на следующие категории:

1. **Арифметические операторы (Arithmetic operators)** - к ним относятся унарные $+$, $-$ и бинарные $+$, $-$, $*$, $/$, $\%$ (остаток от деления).

```
System.out.println(17%3);
```

 Ответ 2

```
System.out.println(3%17);
```

 Ответ 3
Если делимое меньше делителя, то оно полностью выпадает в остаток.

```
System.out.println(17/3);
```

 Ответ 5

```
System.out.println(17./3);
```

 Ответ 17
стало double 5.666666666666667

Операция остаток от деления $\%$ применяется в том случае когда нужно получить разряды числа или ограничить числовой диапазон сверху (например до 100 или до 10).

2. **Оператор присваивания (Assignment Operator)** = Переменная слева присваивает значение переменной (выражения) справа.

$l\text{-value} = r\text{-value};$

$l\text{-value}$ - это переменная слева, то куда мы хотим записать. $r\text{-value}$ - выражение справа, то что мы хотим записать. В простейшем случае выражение справа состоит из одной переменной или константы. Но может быть и сколько угодно сложным. `int a = 2;`

Присвоить - значит записать, сохранить в память, заполнить, дать значение.

Д/з выучить лекцию начиная с Инициализации. Практика: 1) Есть переменная введенная с клавиатуры, необходимо вывести на экран, поменять их значения местами и снова вывести и на экран. 2) С клавиатуры вводится число, нужно

определить, является ли это число палиндромом. Палиндром - это число которое читается одинаково в обоих направлениях. 12321, 1221, 121.

3. Инкремент и декремент (++ и --)

Инкремент increment (++) - это унарный оператор который увеличивает значение переменной на единицу.

Декремент Decrement (--) - уменьшает значение переменной на 1.

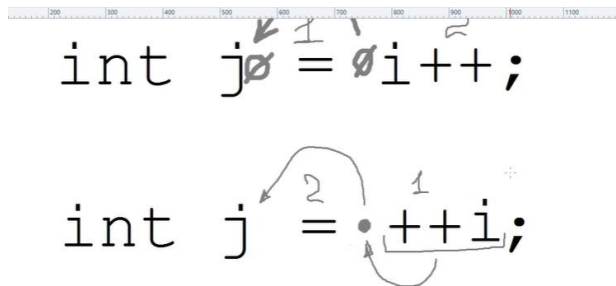
У инкремента и декремента есть две формы записи **префиксная и постфиксная** (суффиксная). В префиксной форме записи оператор пишется перед операндом. А в постфиксной после операнда.

```
int i=0;
++i; //префикс инкремент
System.out.println(i);
i++; //постфиксный инкремент
System.out.println(i);
--i; //префиксный декремент
System.out.println(i);
i--; //постфиксный (суффикс) декремент
System.out.println(i);
```

Префиксная и постфиксная запись инкремента и декремента отличаются приоритетом по сравнению с другими операторами у префиксной формы записи приоритет выше чем у других операторов, а у постфиксной формы записи ниже чем у других операторов. Можно сказать что у префиксной формы записи самый высокий приоритет, а постфиксной самый низкий приоритет.

```
int i=0;
int j = i++; //это выражение сложное потому что тут больше одного оператора,
операторы выполняются в порядке своего приоритета. вначале идет = потом уже ++
System.out.println(i);
System.out.println(j);
```

1
0



```
int i=0;
int j = ++i; //Префиксная форма записи, тут наивысший приоритет. Выражение сложное
4+1+ (i*2)=4+1+ ++(3*2)=4+1+7=12 выполняется в приоритете операторов идет вначале
++ потом =. он станет операндом справа для присваивания и вторая операция
попадает в переменную j
```

```
System.out.println(i);
System.out.println(j);
```

1
1

Используя только оператор инкремента, измени значение в переменной six, чтобы на экран вывелась цифра 9.

```

public static void main(String[] args) {
    int six = 6;
    six ++;
    six ++;
    six ++;
    System.out.println(six);
}

```

Используя только оператор декремента, измени значение переменной toothCounter, чтобы на экран вывелось число 23.

```

public class Solution {
    public static void main(String[] args) {
        int toothCounter = 32;
        toothCounter--;
        toothCounter--;
        toothCounter--;
        toothCounter--;
        toothCounter--;
        toothCounter--;
        toothCounter--;
        toothCounter--;
        System.out.println(toothCounter);
    }
}

```

Допустим, имеется целочисленная переменная с именем counter. При выполнении команды System.out.println(counter++); программа сначала выведет исходное значение counter, а потом увеличит counter на 1. Другими словами, операции выполняются в следующем порядке:

```
System.out.println(counter);
```

```
counter = counter + 1;
```

С другой стороны, при выполнении команды System.out.println(++counter); программа увеличит counter на 1 до того, как вывести новое значение counter. Иначе говоря, операции выполняются в порядке

```
counter = counter + 1;
```

```
System.out.println(counter);
```

4. Составные присваивания (Compound assignments)

assign - присвоить

assignment - присваивание

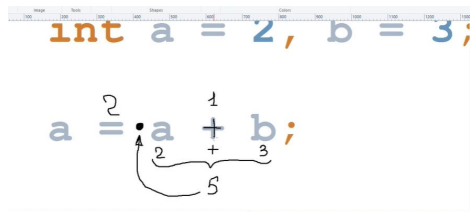
Если переменную надо увеличить или уменьшить не на единицу, а на другое значение или в несколько раз, то для этих целей используются составные присваивания. Это комбинации арифметических операторов с оператором присваивания. составные присваивания позволяют упростить выражения вида

```
int a = 2 , b = 3; // вначале идет + потом уже присваивание =
```

```
System.out.printf("%d \t %d\n", a ,b);
```

```
a = a + b; // 5
```

```
System.out.printf("%d \t %d\n", a ,b);
```



Все операторы возвращают значения

$a += b$ & $a = a + b$; приводят к одному и тому же результату. $a -= b$ & $a = a - b$;
 $a *= b$ & $a = a * b$; и $a /= b$ & $a = a / b$; $a \% = b$ & $a = a \% b$;

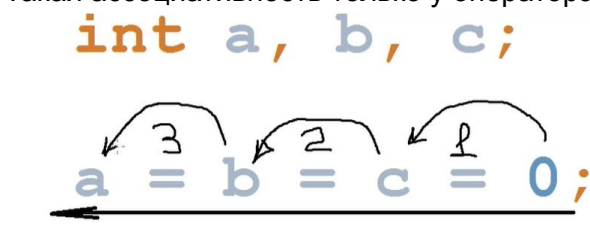
$a += b$;	$a = a + b$;
$a -= b$;	$a = a - b$;
$a *= b$;	$a = a * b$;
$a /= b$;	$a = a / b$;
$a \% = b$;	$a = a \% b$;

Допустим, имеется переменная x с исходным значением 10. Если вы хотите увеличить x на 2, можно использовать запись вида $x = x + 2$; Программа сначала вычисляет значение в правой части ($x + 2$), а затем присваивает ответ переменной в левой части. Таким образом переменная x становится равной 12. Вместо записи $x = x + 2$ можно использовать команду $x += 2$ для выполнения той же операции. Оператор $+=$ представляет собой сокращенную запись, которая объединяет оператор присваивания с оператором сложения. Таким образом, $x += 2$ означает просто $x = x + 2$.

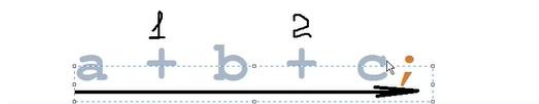
У оператора присваивания очень низкий приоритет и ассоциативность справа налево. Ассоциативность включается когда выражение состоит из операторов с одинаковым приоритетом. В таком случае выражение читается слева направо или справа налево. У всех операторов ассоциативность слева направо кроме присваивания. У операторов присваивания ассоциативность справа налево.

```
int a, b, c;
a = b = c = 7;
System.out.printf("%d\t%d\t%d\t\n", a, b, c);
```

Такая ассоциативность только у оператора присваивания



если + то все будет по цепочке



В случае присваивания ассоциативность имеет преимущество над приоритетом.

5. Операторы сравнения (Comparison Operators)

Math	Java
=	==
≠	!=
>	>
<	<
≥	>=
≤	<=

Все операторы сравнения возвращают **true & false** - это значение **типа bool**.

Операторы сравнения чаще всего используются для написания условий в управляющих структурах. **УСЛОВИЕ (CONDITION) - ЭТО СРАВНЕНИЕ**

Условие состоящее из одной операции сравнения называют **простым**. Простые условия можно объединять в сложное при помощи логических операторов.

6. **Логические операторы** (Logical operators). Существует всего три логических оператора ! - not, || - or, && - and.

! (not) унарный оператор, который отрицает условие. **!true==false**; вернет true

Все остальные логические операторы бинарные. Например для покупки сигарет можно использовать паспорт водительские и тд.

|| (or) результатом сложного условия будет **true**, если результат хотя бы одного простого условия **true**. Например если вы не сделали справку (а паспорт и псих вы сделали), то вам не выдадут водительские права, если хотя бы одно условие не соблюдается. Оператор OR || напоминает арифметическое сложение 1 и 0. $0+0+1+0+1+0 = 1$ - true это все что не 0. $1+1+0+1...$

&& - and результатом сложного условия будет **false**, если результат хотя бы одного простого условия **false**. Например если на улице дождь то сидим дома, если не дождик то идем гулять. $1*1*1*1*1*0 = 0$ - это false $0*1*1*1*1*1....$

```
int t=20;
boolean rain = true;
if (t>10 && !rain) // not rain
{
    System.out.printf("идем гулять");
}
else {
    System.out.printf("сидим дома");
}
```

Некоторые операторы изменяют свои операнды, а некоторые этого не делают. Но абсолютно все операторы возвращают значения. И в выражении это значение временно безымянным объектом. Временные безымянные выражения существуют только в пределах одного выражения. Как раз благодаря возвращаемому значению, операторы являются операторами и они могут быть частью выражения.

Д/з 1) Выучить теорию. 2) Без использования компилятора вычислить следующие выражения:

```
1.
int i=3;
i = ++i + ++i;
sout(i);
```

2.

```
int i=3;
i = i++ + ++i;
sout(i);
```
3.

```
int i=3;
i = i++ + 1 + ++i*2;
sout(i);
```
4.

```
int i=3;
i = i++ + 1 + ++i*=2;
sout(i);
```
5.

```
int i=3;
i += i++ + 1 + ++i*=2;
sout(i);
```
6.

```
int i=3;
i *= i++ + 1 + ++i+=2;
sout(i);
```

3) Бонусное задание палиндром: аргентина манит негра

Н. Управляющие структуры (20.07.2023)

Часто возникает необходимость сделать выбор того какая часть кода будет выполнена или же многократно выполнить определенную часть кода. Для этого в любом языке программирования есть **управляющие структуры**. Они делятся на две категории: **ветвление и циклы**.

К ветвлениям относятся конструкция if... else.... switch

К циклам относятся while ,do... while, for

IF-ELSE

Конструкция if-else позволяет выполнить один из двух вариантов кода в зависимости от условия. if переводится как «если», а else как «иначе». В конструкции if / else следующий синтаксис:

```

If (Condition) {
    ...;
    code1;
    ....;
}
else {
    ...;
    code2;
    ....;
}

```

Синтаксис конструкции if-else-if:

Condition - это условие, а условие это одно или несколько операций сравнения. Любое условие возвращает true & false. Если условие верно true, то выполняется code 1 при этом code2 и блок else игнорируется. Если условие вернуло false то code 1 игнорируется и выполнение выходит за пределы блока if. В таком случае выполняется блок else и code2 если они есть. Если же их нет, то продолжается обычное выполнение программы. Блок else и code2 являются необязательными элементами конструкции if. При необходимости конструкции if / else можно объединять в цепочку следующим образом:

```

if (Condition_1) {
    ...;
    code1;
    ....;
}
else if (Condition_2) {
    ...;
    code2;
    ....;
}
....
....
else if (Condition_N) {
    ...;
    code_N;
    ....;
}
else {
    default_code;
}

```

В такой цепочке последовательно проверяются условия Condition_1, Condition_2 Condition_N. Если какое то условие вернуло true выполняется соответствующий ему код. После выполнения данного кода происходит выход за пределы всей цепочки, т.е. все условия которые находятся ниже даже не проверяются. Поэтому при написании цепочки if/else очень важен порядок написания условий, поскольку в некоторых случаях некоторые условия даже не будут проверены и их код никогда не будет выполнен. Если все условия вернули false, то будет выполнен default code из последнего else, если он есть.

```

import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner kb= new Scanner(System.in);
        System.out.print("Введите температуру воздуха: ");
        int t = kb.nextInt();
        if (t>70) {
            System.out.println("Вы на другой планете");
        }
        else if (t>50) {
            System.out.println("Вы на экваторе");
        }
        else if (t>35) {
            System.out.println("Очень жарко");
        }
        else if (t>25) {
            System.out.println("Просто жарко");
        }
        else if (t>15) {
            System.out.println("Тепло");
        }
        else if (t>0) {
            System.out.println("Холодно");
        }
        else {

```

```
System.out.println("Мороз");
}}}
```

если написать так, то выполнится сразу 0, потому что например 55 больше 0.

```
import java.util.Scanner;
public class Main {
public static void main(String[] args) {
Scanner kb= new Scanner(System.in);
System.out.print("Введите температуру воздуха: ");
int t = kb.nextInt();
if (t>0) {
System.out.println("Холодно");
}
else if (t>50) {
System.out.println("Вы на экваторе");
}
else if (t>35) {
System.out.println("Очень жарко");
}
else if (t>25) {
System.out.println("Просто жарко");
}
else if (t>15) {
System.out.println("Тепло");
}
else {
System.out.println("Мороз");
}}}
```

если убрать else то конструкция не будет единой и они не зависят друг от друга, проверятся все условия

```
import java.util.Scanner;
public class Main {
public static void main(String[] args) {
Scanner kb= new Scanner(System.in);
System.out.print("Введите температуру воздуха: ");
int t = kb.nextInt();
if (t>70) {
System.out.println("Вы на другой планете");
}
if (t>50) {
System.out.println("Вы на экваторе");
}
if (t>35) {
System.out.println("Очень жарко");
}
if (t>25) {
System.out.println("Просто жарко");
}
if (t>15) {
System.out.println("Тепло");
}
}
```



```
if (t>0) {  
    System.out.println("Холодно");  
}  
else {  
    System.out.println("Мороз");  
}}}
```

Задача 2 Есть диапазон значений от 0 до 10

```
System.out.print("Введите число: ");  
int n = kb.nextInt();  
if (n>0 && n<10) {  
    System.out.println("Вы попали");  
}  
else {  
    System.out.println("Вы промахнулись");  
}
```

Если поставить || всегда вернет true

Если if (n<0 || n>10) всегда вернет false

```
if (age > 18 && age < 50) {  
    System.out.println("Welcome!");  
}
```

```
class Demo {  
    public static void main(String[] args) {  
        int age = 25;  
        int height = 100;  
        if (age > 18 || height > 150) {  
            System.out.println("Welcome!");  
        }  
    }  
}
```

```
int age = 25;  
if(!(age > 18)) { // !(возраст > 18) читается как "если возраст НЕ превышает 18".  
    System.out.println("Too Young");  
} else {  
    System.out.println("Welcome");  
}
```

```
String country = "US";  
int age = 42;  
if((country == "US" || country == "GB") && (age > 0 && age < 100)) {  
    System.out.println("Allowed");  
}
```

```
import java.util.Scanner;  
public class Program {  
    public static void main(String[] args) {
```

```

Scanner sc = new Scanner(System.in);
int age = sc.nextInt();
if(age > 0 && age < 11) {
    System.out.println("Child");
}
else if(age > 11 && age < 17) {
    System.out.println("Teen");
}
else if(age > 17 && age < 64){
    System.out.println("Adult");
}
}
}

```

Ввести с клавиатуры температуру на улице. Если температура меньше 0, вывести надпись "на улице холодно", иначе - вывести надпись "на улице тепло".

```

public static void main(String[] args) {
    String cold = "на улице холодно";
    String warm = "на улице тепло";
    Scanner scan = new Scanner(System.in);
    int temperature = scan.nextInt();
    if (temperature < 0)
        System.out.println(cold);
    else System.out.println(warm);
}

```

Ввести с клавиатуры имя и возраст. Если возраст в пределах 18-28 (включительно), то вывести надпись "Имя, явитесь в военкомат", где Имя - это имя, введенное ранее с клавиатуры

```

public static void main(String[] args) {
    String militaryCommissar = ", явитесь в военкомат";
    Scanner scan = new Scanner(System.in);
    String name = scan.nextLine();
    int age = scan.nextInt();
    if (age >=18 && age <=28)
        System.out.println(name + militaryCommissar);
    else {}
}

```

```

int age = 17;
if (age < 18)
{
    System.out.println("Ты еще ребенок");
    System.out.println("Не спорь со взрослыми");
}
else
{
    System.out.println("Вы уже взрослый");
    System.out.println("Ну и молодежь пошла");
}

```

Если команду(ы) нужно выполнять, только если условие истинно и нет команд, которые нужно выполнять, когда условие ложно, нужно использовать сокращенную запись оператора if — без блока else. Она имеет вид:

if (условие)
команда1;

```
int age = 18;
if (age == 18)
{
    System.out.println("Явитесь в военкомат");
}
else
{
}
```

```
int temperature = 9;

if (temperature > 20)
    System.out.println("надеть рубашку");
else // тут температура меньше (или равна) 20
{
    if (temperature > 10)
        System.out.println("надеть свитер");
    else // тут температура меньше (или равна) 10
    {
        if (temperature > 0)
            System.out.println("надеть плащ");
        else // тут температура меньше 0
            System.out.println("надеть пальто");
    }
}
```

или

```
int temperature = 9;

if (temperature > 20)
    System.out.println("надеть рубашку");
else // тут температура меньше (или равна) 20
    if (temperature > 10)
        System.out.println("надеть свитер");
    else // тут температура меньше (или равна) 10
        if (temperature > 0)
            System.out.println("надеть плащ");
        else // тут температура меньше 0
            System.out.println("надеть пальто");
```

или так

```
int temperature = 9;

if (temperature > 20)
```

```
System.out.println("надеть рубашку");
else if (temperature > 10) // тут температура меньше (или равна) 20
    System.out.println("надеть свитер");
else if (temperature > 0) // тут температура меньше (или равна) 10
    System.out.println("надеть плащ");
else // тут температура меньше 0
    System.out.println("надеть пальто");
```

Важный момент: Если в конструкции if-else не расставлены фигурные скобки, else относится к предыдущему (ближайшему к нему) if-у.

```
int age = 65;
if (age < 60)
    if (age > 20)
        System.out.println("Надо работать");
else
    System.out.println("Можно не работать");
```

```
int age = 65;

if (age < 60)
{
    if (age > 20)
        System.out.println("Надо работать");
    else
        System.out.println("Можно не работать");
}
```

Если смотреть на код в левой части, кажется, что на экран будет выведена надпись «Можно не работать». Однако это не так. На самом деле else и команда «Можно не работать» относятся ко второму (ближайшему) if-у.

В коде справа красным выделены связанные if и else. Также специально расставлены скобки, однозначно показывающие порядок выполнения действий. Надпись Можно не работать никогда не выводится на экран при age больше 60

Есть программа, которая принимает на вход возраст и определяет, нужно в школу или уже пора в институт. Но она работает неправильно. Например, пятилетнего ребенка отправляют в институт! Исправить программу несложно: достаточно в нужном месте поставить фигурные скобки.

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int age = scanner.nextInt();
    if (age < 18)
    {
        if (age >= 6)
            System.out.println("нужно ходить в школу"); }
    else
        System.out.println("пора в институт");
}
```

```

public static void main(String[] args)
{
    Scanner console = new Scanner(System.in); // создаем объект Scanner
    int a = console.nextInt(); // читаем с клавиатуры первое число
    int b = console.nextInt(); // читаем с клавиатуры второе число
    if (a < b)           // если a меньше b
        System.out.println(a); // выводим на экран a
    else                 // иначе
        System.out.println(b); // выводим на экран b
}

```

```

// Демонстрация использования цепочки if-else-if.
class Ladder {
public static void main(String[] args) {
    int x;
    for(x=0; x<6; x++) {
        if(x==1)
            System.out.println("Значение x равно 1");
        else if(x==2)
            System.out.println("Значение x равно 2");
        else if(x==3)
            System.out.println("Значение x равно 3");
        else if(x==4)
            System.out.println("Значение x равно 4");
        else
            System.out.println("Значение x не находится между 1 и 4"); } } }

```

Д/з8 1) Написать калькулятор, который обрабатывает выражение вида 2+3
получает результат 2+3 =5;

```

import java.util.Scanner;
public class Main {
public static void main(String[] args) {
    /*
    int a;
    int b;
    int c;
    char s; // знак операции

    Scanner kb = new Scanner(System.in);

    System.out.print("Введите 1 число: ");
    a = kb.nextInt();
    System.out.print("Введите операцию: ");
    s = kb.next().charAt(0);
    System.out.print("Введите 2 число: ");
    b = kb.nextInt();
    if (s == '+') {
        c=a+b;
    }
    else if (s == '-') {
        c=a-b;
    }
}

```

```

else if (s == '*') {
c=a*b;
}
else if (s == '/') {
c=a/b;
}
else {
c= Integer.MIN_VALUE;
System.out.println("Вы ошиблись с оператором");
}
System.out.printf("%d %c %d = %d", a, s, b, c);
*/
Scanner kb = new Scanner(System.in);
System.out.print("Введите арифметическое выражение: ");
String expression = kb.nextLine();

String[] numbers= expression.split("[\\+\\-\\*\\/]");
System.out.println("Expression:" + expression);
System.out.println(numbers[0]);
System.out.println(numbers[1]);

int a = Integer.parseInt(numbers[0]);
int b = Integer.parseInt(numbers[1]);
int c;

if (expression.indexOf('+') >0 ) {
c=a+b;
System.out.printf("%d + %d = %d", a, b, c);
}
else if (expression.indexOf('-') >0 ) {
c=a-b;
System.out.printf("%d - %d = %d", a, b, c);
}
else if (expression.indexOf('*') >0 ) {
c=a*b;
System.out.printf("%d * %d = %d", a, b, c);
}
else if (expression.indexOf('/') >0 ) {
c=a/b;
System.out.printf("%d / %d = %d", a, b, c);
}
else {
c= Integer.MIN_VALUE;
System.out.println("Вы ошиблись с оператором");
}
}
}
}

```

2) Написать программу для тестирования на 5 вопросах, по 4 варианта ответа на каждый. В конце тестирования программа должна выдать количество правильных ответов.

```

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int a = 0;
        int b = 0;
        int c = 0;
        int d = 0;
        int e = 0;
        Scanner kb = new Scanner(System.in);
        System.out.println ("Вопрос 1) Сколько будет 2+3 ");
        System.out.println (" 4 ");
        System.out.println (" 0 ");
        System.out.println (" 5 ");
        System.out.println (" 0 ");
        a = kb.nextInt();
        System.out.println ("Вопрос 2) Сколько будет 9+0 ");
        System.out.println (" 40 ");
        System.out.println (" 10 ");
        System.out.println (" 9 ");
        System.out.println (" 40 ");
        b = kb.nextInt();
        System.out.println ("Вопрос 3) Сколько будет 8-1 ");
        System.out.println (" 4 ");
        System.out.println (" 0 ");
        System.out.println (" 7 ");
        System.out.println (" 9 ");
        c = kb.nextInt();
        System.out.println ("Вопрос 4) Сколько будет 15+3 ");
        System.out.println (" 6 ");
        System.out.println (" 7 ");
        System.out.println (" 2 ");
        System.out.println (" 18 ");
        d = kb.nextInt();
        System.out.println ("Вопрос 5) Сколько будет 1+1 ");
        System.out.println (" 56 ");
        System.out.println (" 70 ");
        System.out.println (" 2 ");
        System.out.println (" 8 ");
        e = kb.nextInt();

        if (a==5 & b==9 & c==7 & d==18 & e==2) {
            System.out.println("У вас 5 правильных ответов");
        } else if
            ((a!=5 & b==9 & c==7 & d==18 & e==2) || (a==5 & b!=9 & c==7 & d==18 & e==2)
            ||(a==5 & b==9 & c!=7 & d==18 & e==2) || (a==5 & b==9 & c==7 & d!=18 & e==2) || (a==5
            & b==9 & c==7 & d==18 & e!=2)) {
            System.out.println("У вас 4 правильных ответа");
        } else if
            ((a!=5 & b!=9 & c==7 & d==18 & e==2) || (a!=5 & b==9 & c!=7 & d==18 & e==2) ||
            (a!=5 & b==9 & c==7 & d!=18 & e==2) ||(a!=5 & b==9 & c==7 & d==18 & e!=2) || (a==5 &
            b!=9 & c!=7 & d==18 & e==2) || (a==5 & b!=9 & c==7 & d!=18 & e==2) || (a==5 & b!=9 &
            c==7 & d==18 & e!=2) || (a==5 & b==9 & c!=7 & d!=18 & e==2) || (a==5 & b==9 & c!=7 &
            d==18 & e!=2) || (a==5 & b==9 & c==7 & d!=18 & e!=2)) {

```

```

        System.out.println("У вас 3 правильных ответа");
    } else if
        ((a!=5 & b!=9 & c!=7 & d==18 & e==2) || (a!=5 & b!=9 & c==7 & d!=18 & e==2) ||
        (a!=5 & b!=9 & c==7 & d==18 & e!=2) || (a!=5 & b==9 & c!=7 & d!=18 & e==2) || (a!=5 &
        b==9 & c!=7 & d==18 & e!=2) || (a!=5 & b==9 & c==7 & d!=18 & e!=2) || (a==5 & b!=9 &
        c!=7 & d!=18 & e==2) || (a==5 & b!=9 & c!=7 & d==18 & e!=2) || (a==5 & b==9 & c!=7 &
        d!=18 & e!=2) || (a==5 & b!=9 & c==7 & d!=18 & e!=2)) {
        System.out.println("У вас 2 правильных ответа");
    } else if
        ((a==5 & b!=9 & c!=7 & d!=18 & e!=2) || (a!=5 & b==9 & c!=7 & d!=18 & e!=2) || (a!=5
        & b!=9 & c==7 & d!=18 & e!=2) || (a!=5 & b!=9 & c!=7 & d==18 & e!=2) || (a!=5 & b!=9 &
        c!=7 & d!=18 & e==2)) {
        System.out.println("У вас 1 правильный ответ");
    } else {
        System.out.println("Правильных ответов 0");
    }
}
}
}

```

3) **Бонус** Вопросы и ответы должны задаваться в случайном порядке Используя метод range. В программе для тестирования обеспечить вывод для вопросов в случайном порядке. варианты ответов и так же должны перемешиваться.

4) Написать калькулятор при помощи switch();

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html>

```

import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        Scanner kb = new Scanner(System.in);
        System.out.print("Введите 1 число: ");
        int a = kb.nextInt();

        System.out.print("введите 2 число:");
        int b = kb.nextInt();

        System.out.print("Введите оператор (+, -, *, /): ");
        char operator = kb.next().charAt(0);

        int c;

        switch(operator)
        {
            case '+':
                c = a + b;
                break;

            case '-':
                c = a - b;
                break;

            case '*':
                c = a * b;
                break;

```



```

        case '/':
            c = a / b;
            break;
        default:
            System.out.printf("Такого оператора нет");
            return;
    }
    System.out.printf("%d "+ operator + " %d = %d", a, b, c);
}
}

```

ИЛИ

```

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int a = 0;
        int b = 0;
        char op;
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите 1 число: ");
        a = kb.nextInt();
        System.out.print("Введите оператор (+,-,*,/): ");
        op = kb.next().charAt(0);
        System.out.print("Введите 2 число: ");
        b = kb.nextInt();

        char c = '+';
        char d = '-';
        char e = '*';
        char j = '/';
        if (op == c) {
            System.out.printf(a + " " + op + " " + b + " = " + (a + b));
        }
        else if (op == d) {
            System.out.printf(a + " " + op + " " + b + " = " + (a - b));
        }
        else if (op == e) {
            System.out.printf(a + " " + op + " " + b + " = " + (a*b));
        }
        else if (op == j) {
            System.out.printf(a + " " + op + " " + b + " = " + (a/b));
        }
        else {
            System.out.println("Вы ошиблись с оператором");
        }
    }
}

```

```

int day = 2;
if(day == 1) {

```

```
System.out.println("Monday");
} else if(day == 2) {
    System.out.println("Tuesday");
} else if(day == 3) {
    System.out.println("Wednesday");
}
```

с оператором switch

```
int day = 2;

switch(day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    case 3:
        System.out.println("Wednesday");
        break;
}
```

Оператор switch

Для упрощения реализации алгоритма множественного выбора в Java добавлен оператор switch. Оператор анализирует выражение в скобках и передает управление одному из сценариев (case). Далее программа выполнит весь код, находящийся в сценарии до конца оператора switch. Значения вариантов в case не должны повторяться.

Синтаксис switch:

```
switch (выражение)
{
    case вариант1:
        операция1;
    case вариант2:
        операция2;
    case вариант3:
        операция3;
    default:
        операция;
}
```

В скобках допустимо наличие значений примитивных типов byte, short, char, int.

Для прерывания сценария в блоке switch можно использовать ключевое слово break.

Ключевое слово default может использоваться в операторе switch в случае, если ни один сценариев не был выполнен. В блоке switch может присутствовать только одно ключевое слово default. Порядок расположения блоков case и default не имеет значения, но принято располагать блоки case по возрастанию значения, а default – в самом конце.

Оператор switch - это удобный способ проверить наличие нескольких значений и запустить код.

- Помните, что за каждым регистром следует значение и двоеточие.
- Для каждого случая требуется оператор break, иначе код других случаев будет продолжать выполняться.
- Регистр по умолчанию может быть использован для запуска кода, если ни один из регистров не совпадает.

switch основывается на КОНКРЕТНОМ ЗНАЧЕНИИ, тогда как в if. может быть любое логическое выражение

```
import java.util.Scanner;

class Main {
public static void main(String[ ] args) {
Scanner kb = new Scanner(System.in);
String color = kb.next();

switch (color){
case "red":
System.out.println("1");
break;
case "green":
System.out.println("2");
break;
case "black":
System.out.println("3");
break;
}}}
```

```
public class SwitchDemo {
    public static void main(String[] args) {

        int month = 8;
        String monthString;
        switch (month) {
            case 1: monthString = "January";
                    break;
            case 2: monthString = "February";
                    break;
            case 3: monthString = "March";
                    break;
            case 4: monthString = "April";
                    break;
            case 5: monthString = "May";
                    break;
            case 6: monthString = "June";
                    break;
            case 7: monthString = "July";
                    break;
            case 8: monthString = "August";
                    break;
            case 9: monthString = "September";
                    break;
            case 10: monthString = "October";
                    break;
        }
```

```
        case 11: monthString = "November";
            break;
        case 12: monthString = "December";
            break;
        default: monthString = "Invalid month";
            break;
    }
    System.out.println(monthString);
}
}
```

используя String

```
public class StringSwitchDemo {

    public static int getMonthNumber(String month) {

        int monthNumber = 0;

        if (month == null) {
            return monthNumber;
        }

        switch (month.toLowerCase()) {
            case "january":
                monthNumber = 1;
                break;
            case "february":
                monthNumber = 2;
                break;
            case "march":
                monthNumber = 3;
                break;
            case "april":
                monthNumber = 4;
                break;
            case "may":
                monthNumber = 5;
                break;
            case "june":
                monthNumber = 6;
                break;
            case "july":
                monthNumber = 7;
                break;
            case "august":
                monthNumber = 8;
                break;
            case "september":
                monthNumber = 9;
                break;
            case "october":
                monthNumber = 10;
                break;
        }
    }
}
```

```

        case "november":
            monthNumber = 11;
            break;
        case "december":
            monthNumber = 12;
            break;
        default:
            monthNumber = 0;
            break;
    }
    return monthNumber;
}

```

```

public static void main(String[] args) {
    String month = "August";

    int returnedMonthNumber =
        StringSwitchDemo.getMonthNumber(month);

    if (returnedMonthNumber == 0) {
        System.out.println("Invalid month");
    } else {
        System.out.println(returnedMonthNumber);
    }
}
}

```

I. Конструкция множественного выбора switch (27.07.2023)

В отличие от If-else которые позволяют выбрать один из двух вариантов кода в зависимости от условия, **switch** позволяет выбрать один из множества вариантов кода в зависимости от значения некоторой переменной. Эту конструкции switch следующий синтаксис:

```

switch (var)
{
    case CONST_1: ... code1....; break;
    case CONST_2: ... code2....; break;
        //.....;
        //.....;
    case CONST_N: ... codeN....; break;
    default: ....code...;
}

```

var - это переменная по значению которой switch выбирает какой вариант кода нужно выполнить. Эту переменную он последовательно сравнивает с константой после ключевого слова case -> CONST_1, CONST_2....CONST_N и при совпадении переходит на соответствующий кейс и выполняет соответствующий код до ключевого слова break. Ключевое слова break прерывает выполнение кода и выходит за пределы конструкции switch. Если ключевое слово break отсутствует, то выполнится код соответствующий следующему case и т.д. до тех пор пока не видится break или не закончится switch. Если переменная var не совпала ни с одной из констант, то выполнится код после метки default если она есть.

Переменная var и константы CONST_1, CONST_2....CONST_N могут быть лишь целочисленного и символьного типа. В последних версиях языка и строкового типа.

```

switch (s)
{
    case '+': c = a + b; System.out.printf("%d + %d = %d", a, b, c); break;
    case '-': c = a - b; System.out.printf("%d - %d = %d", a, b, c); break;
    case '*': c = a * b; System.out.printf("%d * %d = %d", a, b, c); break;
    case '/': c = a / b; System.out.printf("%d / %d = %d", a, b, c); break;
    default:
        System.out.printf("No operation");
}

```

или

```

switch (s)
{
    case '+': System.out.printf("%d + %d = %d", a, b, a+b); break;
    case '-': System.out.printf("%d - %d = %d", a, b, a-b); break;
    case '*': System.out.printf("%d * %d = %d", a, b, a*b); break;
    case '/': System.out.printf("%d / %d = %d", a, b, a/b); break;
    default:
        System.out.printf("No operation");
}

```

Программа выводит меню с названиями управляющих операторов и ожидает выбора одного из них, после которого отобразится синтаксис оператора. В первой версии программы справочная информация доступна только для оператора if и традиционного оператора switch. Сведения для других управляющих операторов будут добавлены в последующих проектах

```

/*
Упражнение 3.1.
Простая справочная система по управляющим операторам Java.
*/
class Help {
    public static void main(String[] args)
        throws java.io.IOException {
        char choice;
        System.out.println("Справка по:");
        System.out.println(" 1. if");
        System.out.println(" 2. switch");
        System.out.print("Выберите вариант: ");
        choice =
            System.out.println("\n");
        switch(choice) {
            case '1':
                System.out.println("Оператор if:\n");
                System.out.println("if(условие) оператор;");
                System.out.println("else оператор;");
                break;
            case '2':
                System.out.println("Традиционный оператор switch:\n");
                System.out.println("switch(выражение) {");
                System.out.println("    case константа:");
                System.out.println("        ");
                System.out.println("        ");
                System.out.println("        // ...");
                System.out.println("    }");

```

```
break;
default:
System.out.print("Выбранный вариант не найден.");
последовательность операторов");
break;");
}}}
```

Ж. Циклы

Цикл (loop) - это управляющая структура которая позволяет многократно выполнить определенный код или же повторить выполнение какого то кода. Например, банковское приложение может перебирать все банковские транзакции и проверять выполнение некоторых условий. Существует три типа циклов: **while (пока не)** - цикл с предусловием, **do...while** цикл с постусловием, **for** - цикл на заданное число раз. У любого цикла есть условие которое представляет собой одно или несколько операций сравнения. **Итерация** - это однократное выполнение тела цикла. **Тело цикла** - это код, который нужно многократно выполнить. Циклы while и do...while имеют следующий синтаксис:

<pre>while (condition) { ; group-of-statements; ; }</pre>	<pre>do { ; group-of-statements; ; } while (condition);</pre>
---	---

Цикл while работает следующим образом:

1. проверяется условие;
2. если условие вернуло true, выполняется тело цикла;
3. происходит возврат в начало и снова проверяется условие и так до тех пор пока условие не вернут false;
4. если условие вернуло false, то происходит выход за пределы цикла.
5. Если члены условия цикла не меняются внутри тела цикла, а условие выполнения цикла истинно, такой цикл будет выполняться бесконечно.

Оператор цикла while очень простой и состоит всего из двух частей: *условие* и *тело цикла*. Тело цикла выполняется снова и снова, пока условие равно true. Общий вид цикла while такой:

```
while (условие)
    команда;
Запись цикла while с одной командой
```

```
while (условие)
{
    блок команд
}
```

Все очень просто. *Команда* или *блок команд* выполняются снова и снова, пока *условие цикла* истинно — равно true. Это работает так: сначала проверяется *условие*, и если оно истинно, выполняется *тело цикла* (*команда* или *блок команд*), затем снова проверяется *условие* и снова выполняется *тело цикла*. И так до тех пор, пока *условие* не станет ложным

```
int n = 5;
while (n > 0)
{
    System.out.println(n);
    n--;
}
```

```
int x = 3;

while(x > 0) {
    System.out.println(x);
    x = x-1;
}
Например, оператор x=x-1; может быть упрощен до x--;
```

Например, позволяет выводить только четные числа от 0 до 10.

```
int x = 0;
while(x<=10) {
    System.out.println(x);
    x = x+2;
}
```

Существует также более короткий способ для $x=x+2$; он может быть записан как $x+=2$;

Например, давайте вычислим сумму чисел от 1 до 100 и выведем ее

```
int sum = 0;
int num = 0;

while(num <= 100) {
    sum += num;
    num++;
}
System.out.println(sum);
```

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите число: ");
        int n = kb.nextInt();
        int i = 1;
        int sum = 0;
        while (i <= n) {
            sum = sum + i;
            i++;
        }
        System.out.println(sum);
    }
}
```


Программа считывает с клавиатуры числа до тех пор, пока вводят именно числа.

```
Scanner console = new Scanner(System.in);
while(console.hasNextInt())
{
    int x = console.nextInt();
}
```

Программа будет вводить строки с клавиатуры, пока не будет введена строка exit. Функция equals() в примере используется для сравнения строк. Если строки равны, функция вернет результат — true, если строки не равны, то вернет false.

```
Scanner console = new Scanner(System.in);
boolean isExit = false;
while (!isExit)
{
    String s = console.nextLine();
    isExit = s.equals("exit");
}
```

Используя цикл while вывести на экран сто раз цитату (переменная quote): «Я никогда не буду работать за копейки. Амиго». Каждое значение вывести с новой строки.

```
public class Solution {
    public static void main(String[] args) {
        String quote = "Я никогда не буду работать за копейки. Амиго";
        int n = 0;
        while (n < 100)
        {
            System.out.println(quote);
            n++;
        }
    }
}
```

Ввести с клавиатуры имя и, используя цикл while, 10 раз вывести: <имя> любит меня (переменная text). Каждый вывод - с новой строки.

```
public class Solution {
    public static void main(String[] args) {
        String text = " любит меня.";
        Scanner scanner = new Scanner(System.in);
        String s1 = scanner.nextLine();

        int n = 0;
        while (n < 10)
        {
            System.out.println(s1 + text);
            n++;
        }
    }
}
```

Напишем программу, в которой нужно вводить с клавиатуры целые числа и считать их сумму, пока пользователь не введёт слово "ENTER". Вывести на экран полученную сумму и завершить программу.

```
public class Solution {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int sum = 0;
        while (true){
            String input = scanner.nextLine();
            if (input.equals("ENTER")) {
                break;
            }
            sum += Integer.parseInt(input);
        }
        System.out.println(sum);
    }
}
```

Давайте напишем программу, которая вводит числа с клавиатуры (пока пользователь вводит что-то похожее на числа) и выводит на экран их сумму. Вот как будет выглядеть код такой программы (приводим только код внутри метода main).

```
Scanner console = new Scanner(System.in);
int sum = 0;
while (console.hasNextInt())
{
    int x = console.nextInt();
    sum = sum + x;
}
System.out.println(sum);
```

Давайте напишем программу, которая вводит числа с клавиатуры (пока пользователь вводит что-то похожее на числа) и выводит на экран их сумму. Вот как будет выглядеть код такой программы (приводим только код внутри метода main). Создаем объект Scanner для чтения данных с консоли. В переменной sum будем хранить сумму чисел. Пока в консоли вводят целые числа считываем очередное число в переменную x. Добавляем x к сумме чисел (переменная sum). Выводим подсчитанную сумму на экран.

```
Scanner console = new Scanner(System.in);
int sum = 0;
while (console.hasNextInt())
{
    int x = console.nextInt();
    sum = sum + x;
}
System.out.println(sum);
```

Вторая программа тоже будет считывать числа с клавиатуры (пока пользователь вводит что-то похожее на числа), но на экран нужно будет вывести наибольшее из введенных чисел. Вот как будет выглядеть код такой программы (приводим только код внутри метода main). Создаем объект Scanner для чтения данных с консоли.

В переменной max будем хранить максимум чисел. Пока в консоли вводят целые числа считываем очередное число в переменную x. Сравниваем x и max. Если x больше max, обновляем максимум. Выводим найденный максимум чисел на экран.

```
Scanner console = new Scanner(System.in);
int max = 0;
while (console.hasNextInt())
{
    int x = console.nextInt();
    if (x > max)
        max = x;
}
System.out.println(max);
```

Программа должна считывать целые числа с клавиатуры до тех пор, пока не будет введено что-то другое: например, строка или символ. Выведи на экран минимальное число из введенных. Если введено несколько таких чисел, необходимо вывести любое из них.

```
public class Solution {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        int min = Integer.MAX_VALUE;
        while (console.hasNextInt())
        {
            int x = console.nextInt();
            if (x <= min )
                min = x;
        }
        System.out.println(min);
    }
}
```

Программа должна считывать целые числа с клавиатуры до тех пор, пока не будет введено что-то другое: например, строка или символ. Выведи на экран максимальное четное число из введенных. Если введено несколько таких чисел, необходимо вывести любое из них.

```
public class Solution {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        int max = Integer.MIN_VALUE;
        while (console.hasNextInt())
        {
            int x = console.nextInt();

            if (max < x && x % 2 == 0)
                max = x;
        }
        System.out.println(max);
    }
}
```

В этой задаче нужно: Считывать целые числа с клавиатуры до тех пор, пока не будет введено что-то другое: например, строка или символ. Должно быть введено минимум два числа. Если введено менее двух целых чисел, то такую ситуацию обрабатывать не нужно, а программа может быть завершена с ошибкой. Вывести на экран второе по величине целое число после минимального из введенных с клавиатуры. Если таких чисел несколько, то необходимо вывести любое из них. Минимальных чисел тоже может быть несколько.

```
public class Solution {
    public static void main(String[] args) {

        Scanner scan=new Scanner(System.in);
        int min=Integer.MAX_VALUE;
        int min2=Integer.MAX_VALUE;

        while (scan.hasNextInt()){
            int a= scan.nextInt();
            if (a<min ){
                min2=min;
                min=a;

            } else if (a<min2 && a!=min) {
                min2=a;
            }
        }System.out.println(min2);
    }
}
```

Далее представлен простой пример, в котором цикл while используется для вывода букв английского алфавита: Переменная ch инициализируется буквой а. При каждом проходе цикла переменная ch выводится и затем инкрементируется. Процесс продолжается до тех пор, пока ch не станет больше z.

```
// Демонстрация применения цикла while.
class WhileDemo {
    public static void main(String[] args) {
        char ch;
        // Вывести буквы английского алфавита, используя цикл while,
        ch = 'a';
        while(ch <= 'z') {
            System.out.print(ch);
            ch++;
        }
    }
}
```

// Вычисление целых степеней числа 2. Обратите внимание, что цикл while выполняется только когда e больше 0. Таким образом, когда значение e равно 0, как в первой итерации цикла for, цикл while пропускается.

```
class Power {
    public static void main(String[] args) {
        int e;
        int result;
```

```

for(int i=0; i < 10; i++) {
    result = 1;
    e = i;
    while(e > 0) {
        result *= 2; e--;
    }
    System.out.println("2 в степени " + i + " равно " + result);
}
}
}

```

Цикл в цикле

Чтобы написать цикл в цикле, нужно в теле первого цикла написать второй. Выглядеть это будет примерно так:

```

while (условие внешнего цикла)
{
    while (условие внутреннего цикла)
    {
        блок команд
    }
}

```

Допустим, мы хотим написать программу, которая бы выводила на экран 4 раза слово Мама. Цикл — это именно то, что нам нужно. И примерно так выглядел бы наш код:

```

int n = 0;
while (n < 4)
{
    System.out.println("Мама");
    n++;
}

```

Мы хотим написать программу, которая бы выводила в одну строку 5 букв А. Для этого нам опять нужен цикл. Вот как будет выглядеть этот код

```

int n = 0;
while (n < 5)
{
    System.out.print("A");
    n++;
}

```

Мы хотим вывести на экран прямоугольник из букв А размером в 4 строки и 5 столбцов. А вот для этого нам уже нужны вложенные циклы. Просто возьмем первый пример, где мы выводим 4 строки и заменим код по выводу одной строки на код из второго примера. Также пришлось добавить команду System.out.println() после внутреннего цикла, т.к он печатает буквы А в одну строку, и после вывода всех букв кто-то должен перенести курсор на новую строчку.

```

int n = 0;
while (n < 4) // 4 строки
{
    int m = 0;
    while (m < 5) // 5 столбцов
    {
        System.out.print("A");
    }
    System.out.println();
}

```

```

        m++;
    }
    System.out.println();
    n++;
}

```

```
public class Solution {
    public static void main(String[] args) {
        int n = 0;
        while (n < 5)
        {

            int m = 0;
            while (m < 10)
            {
                System.out.print("Q");
                m++;
            }

            System.out.println();
            n++;
        }
    }
}
```

Вложенные циклы используются для решения самых разных задач программирования и являются неотъемлемой частью программирования.

```
public class Solution {
    public static void main(String[] args) {
        //напишите тут ваш код
        System.out.print("BBBBBBBBBBBBBBBBBBBB");
        System.out.println();
        int height = 0;
        while (height < 8)
        {
            int width = 0;
            while (width < 20)
            {
                width++;
            }
            System.out.println("Б          Б");
            height++;
        }
        System.out.print("BBBBBBBBBBBBBBBBBBBB");    }}

```

Внешний цикл в программе выполняется для значений *i* от 2 до 100. Во внутреннем цикле последовательно проверяются все числа от 2 до *i* и выводятся те из них, которые обеспечивают деление значения *i* нацело.

```
/* Использование вложенного цикла для нахождения множителей чисел от 2 до 100.
*/
class FindFac {
public static void main(String[] args) {
for(int i=2; i <= 100; i++) {
System.out.print("Множители " + i + ": ");
for(int j = 2; j < i; j++)
if((i%j) == 0) System.out.print(j + " ");
System.out.println();
}
}
}
```

Цикл do-while (цикл с постусловием) работает следующим образом:

- 1) выполняется тело цикла;
- 2) проверяется условие;
- 3) если условие вернуло true происходит возврат в начало (do) и снова выполняется тело цикла и так до тех пор пока условие не вернет false;
- 4) если условие вернуло false, то происходит выход за пределы цикла.

```
do {
    последовательность операторов
} while ( условие );
```

```
class Demo {
    public static void main(String[] args) {
        int x = 1;
        do {
            System.out.println(x);
            x++;
        } while(x < 5);
    }
}
```

Циклы while и do...while отличаются тем, что do...while в любом случае выполнится хотя бы один раз независимо от условия (Разница между while и do-while заключается в том, что do-while гарантированно запустится хотя бы один раз, даже при ложном условии) Если условие цикла while при первой же проверке вернет false, то тело цикла ни разу не выполниться. Цикл while вначале думает, а потом делает. Цикл do...while делает, а потом думает. do...while обычно применяют в том случае когда переменная присутствующая в условии инициализируется в теле цикла. Иногда это позволяет сократить объем кода

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        int i=0; //счетчик цикла
        System.out.print("Введите количество итераций: ");
        int n = kb.nextInt(); // количество итераций вводимых с клавиатуры
```

```

while(i < n)
{
    System.out.println("%d - Hello".formatted(i)); // добавили счетчик
    i++; // или можно так System.out.println("%d - Hello".formatted(i++));
    System.out.println("%d - Hello".formatted(++i)); // так будет вначале переменная а
    //потому уже выводиться на экран
}
}
}

```

```

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        int i = 0; //счетчик цикла
        System.out.print("Введите количество итераций: ");
        int n = kb.nextInt(); // количество итераций вводимых с клавиатуры
        while(i < n)
        {
            System.out.println("%d - Hello".formatted(i)); //добавили счетчик
            formatted
            i++
            // или можно так System.out.println("%d - Hello".formatted(i++));
            //System.out.println("%d - Hello".formatted(++i)); так будет вначале переменная а
            //потому уже выводиться на экран
        }
    }
}

```

```

import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        String integer;
        do {
            System.out.println("Введите целое число");
            integer= kb.next();
            if (integer.indexOf(',') != -1 || integer.indexOf('.') != -1 ) System.out.println("Вы ввели
            дробное число, нужно ввести целое число");
        }while(integer.indexOf(',') != -1 || integer.indexOf('.') != -1 ); //возвращается его индекс
        System.out.println();
    }
}

```

Цикл do-while

Последней разновидностью циклов Java является do-while. В отличие от циклов for и while, где условие проверяется в начале цикла, в цикле do-while условие проверяется в конце цикла. Таким образом, цикл do-while будет выполняться хотя бы один раз.


```

do
    команда;
while (условие);

или

do
{
    БЛОК КОМАНД
}
while (условие);

```

Но если в цикле while последовательность выполнения будет такой: *условие, тело цикла, условие, тело цикла, условие, тело цикла, ...* То в do-while она будет немного другой: *тело цикла, условие, тело цикла, условие, тело цикла, ...* Фактически разница между while и do while только в том, что *тело цикла* в цикле do-while выполняется *как минимум один раз*.

В следующей программе цикл выполняется до тех пор, пока пользователь не введет букву q:

```

// Демонстрация использования цикла do-while.
class DWDemo {
public static void main(String[] args)
throws java.io.IOException {
char ch;
do {
System.out.print("Нажмите клавишу и затем ENTER: ");
ch = (char) System.in.read(); // получить символ
} while(ch != 'q');
}
}

```

С использованием цикла do-while можно дополнительно улучшить программу игры в угадывание буквы. На этот раз цикл повторяется до тех пор, пока буква не будет угадана.

```

// Игра в угадывание буквы , версия 4.
class Guess4 {
public static void main(String[] args)
throws java.io.IOException {
char ch, ignore, answer = 'K';
do {
System.out.println("Задумана буква между A и Z.");
System.out.print("Попробуйте ее угадать: ");
// Прочитать символ.
ch = (char) System.in.read();
// Отбросить все остальные символы из буфера ввода,
do {
ignore = (char) System.in.read();
} while(ignore != '\n');
if(ch == answer) System.out.println("Правильно");
else {

```

```

System.out.print("...Увы , не угадали. Задуманная буква находится ");
if(ch < answer) System.out.println("дальше по алфавиту.");
else System.out.println("ближе по алфавиту.");
System.out.println("Повторите попытку!\n");
}
} while(answer != ch);
}
}

```

```

/*
Упражнение 3.2.
Улучшенная справочная система по управляющим операторам Java,
в которой используется цикл do-while для обработки выбора
варианта в меню.
*/
class He1p2 {
public static void main(String[] args)
throws java.io.IOException {
char choice, ignore;
do {
System.out.println("Справка no:");
System.out.println("
System.out.println("
System.out.println("
System.out.println("
System.out.println("
System.out.print("Выберите вариант:
(char) System.in.read();
1. if");
2. switch");
3. for");
4. while");
5. do-while\n");
");
choice
do {
ignore = (char) System.in.read();
} while(ignore != '\n * ');
} while( choice < '1'|choice > '5');
System.out.println("\n");
switch(choice) {
case '1':
System.out.println("Оператор if:\n");
System.out.println("if(условие) оператор;");
System.out.println("else оператор;");
break;
case '2':
System.out.println("Традиционный оператор switch:\n");
System.out.println("switch(выражение) {");
System.out.println("
System.out.println("
System.out.println("
System.out.println(" // ...");

```

```

System.out.println("}");
break;
case '3':
System.out.println("Цикл for:\n");
System.out.print("for(инициализация; условие; итерация)");
System.out.println(" оператор;");
break;
case '4':
System.out.println("Цикл while:\n");
System.out.println("while(условие) оператор;");
break;
case '5':
System.out.println("Цикл do-while:\n");
System.out.println("do {");
System.out.println(" оператор;");
System.out.println("} while (условие);");
break;
case константа:");
последовательность операторов");
break;"); } }

```

Программа вводит строки с клавиатуры, пока не введено слово **exit**

while	do while
<pre> String s; while (true) { s = console.nextLine(); if (s.equals("exit")) break; } </pre>	<pre> String s; do { s = console.nextLine(); } while (!s.equals("exit")); </pre>

В этой задаче нужно: Ввести с клавиатуры строку и число number, которое больше 0 и меньше 5. Вывести на экран строку number раз с помощью цикла do-while. Каждое значение нужно вывести с новой строки.

```

package com.javarush.task.pro.task04.task0414;
import java.util.Scanner; /* Хорошего не бывает много*/

public class Solution {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        String str = kb.nextLine();
        int num = kb.nextInt();
        int i=0;

        if (num<=0 || num>=5) {
            System.out.println(str);
        }
        else do {
            for (i=0;i<num;i++)
                System.out.println(str);
        }
    }
}

```

```
}  
while (i!=num);  }}
```

Цикл for (выполнить цикл n-раз) - цикл на заданное число итераций. Его применяют в том случае когда знают количество повторений. Цикл for имеет следующий синтаксис:

```
for (counter; condition; expression)  
{  
    .....;  
    group-of-statements;  
    .....;  
}
```

counter - это счетчик цикла. **Счетчик** - это переменная которая считает сколько на выполнялся или сколько раз будет выполнен цикл. Эту переменную можно объявить как в самом цикле так и перед ним. Если счетчик объявлен в самом цикле for он является локальным для этого for, поскольку существует только в его области видимости и не существует за ее пределы. Область видимости for ограничивается его круглыми и фигурными скобками. Локальный счетчик удаляется из памяти по завершении всех итерации. Если счетчик глобальный то есть объявлен перед for то он будет существовать и после него, а for лишь изменит глобальный счетчик. Выражение **counter обрабатывает один раз перед первой итерацией**.

condition - это условие продолжение или завершения цикла. В этом условии как правило счетчик сравнивается с каким либо значением. И если условие вернуло true, то выполняется тело цикла, в противном случае (если false) происходит выход за пределы цикла. Условие проверяется перед каждой итерацией, если же при первой проверке оно вернуло false, то тело цикла ни разу не выполнится.

expression - это выражение которое изменяет счетчик. Как правило тут пишут инкремент или декремент счетчика, хотя сюда можно писать все что угодно.

expression - обрабатывает после каждой итерации после чего снова проверяется условие.

group-of-statements - это тело цикла. В теле цикла можно писать все что угодно.

Если коротко: **counter** выполняется один раз, когда мы входим в цикл и инициализируем переменную. **condition** - это условие цикла. **expression** запускается каждый раз, когда выполняется цикл.

Цикл for лучше всего подходит, когда мы знаем, сколько раз нам нужно запустить цикл.

Цикл for как правило пишут так:

```
int n = 10;  
for (int i = 0; i < n; i++)  
{  
    sout(i + "\t");  
}
```

```
/*
```

Демонстрация использования цикла for.

Назовите этот файл ForDemo.java.

```
*/
```

```
class ForDemo {
```

```
public static void main(String[] args) {
```

```
int count;
```

```
for(count = 0; count < 5; count = count+1) // или так for(count = 0; count < 5; count++)
```

```
System.out.println("Значение count: " + count);
System.out.println("Готово!");
Этот цикл выполняется пять раз }}
```

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите количество итераций: ");
        int n=kb.nextInt();
        for (int i = 0; i < n; i++)
        {
            System.out.print(i + "\t");
        }
        System.out.println();
    }
}
```

В приведенном ниже примере выводятся только четные значения от 0 до 10

```
for(int x=0; x<=10; x+=2) {
    System.out.println(x);
}
```

```
int x = 1;
while(x < 10) {
    System.out.println(x);
    if(x == 4) {
        break;
    }
    x++;
}
```

Оператор break может быть использован для остановки цикла.

```
для for будет так
for(int x=1; x<10; x++) {
    System.out.println(x);
    if(x == 4) {
        break;
    }
}
```

Другим управляющим оператором является continue. Это заставляет цикл пропускать текущую итерацию и переходить к следующей.

```
for(int x=10; x<=40; x+=10) {
    if(x == 30) {
        continue; // код пропускает значение 30, как указано в инструкции continue.
    }
    System.out.println(x);
}
```

Билетная система авиакомпании должна рассчитать общую стоимость всех приобретенных билетов. Билеты для детей в возрасте до 3 лет предоставляются бесплатно. Мы можем использовать цикл while для перебора списка пассажиров и расчета общей стоимости их билетов. Здесь оператор continue может быть использован для пропуска дочерних элементов.

```
/*
Упражнение 1.2.
Эта программа отображает таблицу преобразований галлонов в литры .
Назовите этот файл GalToLitTable.java.
*/
class GalToLitTable {
public static void main(String[] args) {
double gallons, liters;
int counter;
counter = 0; //Установить счетчик строк сначала в ноль
for(gallons = 1; gallons <= 100; gallons++) {
liters = gallons * 3.7854; // преобразование в литры
System.out.println(gallons + " галлонов соответствует " +
liters + " литрам.");
counter++; //Увеличивать счетчик строк на каждой итерации
// После каждой 10-й строки вывести пустую строку,
if(counter == 10) { — System.out.println(); // Если значение счетчика строк равно 10,
вывести пустую строку
counter = 0; // сброс счетчика строк
}
}
}
}
```

```
for (команда1; условие; команда2)
{
    БЛОК КОМАНД
}
```

Вариант 1	Вариант 2
<pre>for (int i = 0; i < 20; i++) { System.out.println(i); }</pre>	<pre>int i = 0; while (i < 20) { System.out.println(i); i++; }</pre>

Цикл for, наверное, самый используемый вариант цикла в Java. Он применяется везде, т.к. это просто более понятная и удобная форма записи цикла while для программистов. Практически любой цикл while можно преобразовать в цикл for.

Напиши программу, в которой с помощью цикла for на экран будут выведены чётные числа от 1 до 15. Каждое значение нужно выводить с новой строки.

```
public class Solution {
    public static void main(String[] args) {
        for (int i = 1; i <= 15; i++) {
```

```
if((i%2)==0) {  
    System.out.println(i);  
}}}
```

В методе main с клавиатуры считывается 3 целых числа: start, end (start <= end), multiple. Допиши программу, чтобы на экран выводилась сумма чисел от start (включительно) до end (не включительно), кратных multiple. Для этого используй цикл for. Подсказка: чтобы перейти к следующей итерации цикла, используй оператор continue.

```
package com.javarush.task.pro.task04.task0412;  
import java.util.Scanner;  
/*  
Сумма кратных чисел  
*/  
public class Solution {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        int start = scanner.nextInt();  
        int end = scanner.nextInt();  
        int multiple = scanner.nextInt();  
  
        int sum = 0;  
        for(int n = start; n < end; n++){  
            if (n%multiple == 0)  
                sum += n;  
            continue;  
        }  
        System.out.println(sum);  
    }  
}
```

Давайте напишем программу, которая вводит с клавиатуры 10 строк и выводит на экран, сколько из этих строк было чисел. Создаем объект Scanner для чтения данных с консоли. В переменной count будем хранить количество чисел. Цикл от 0 до 10 (не включая 10). Если в консоли ввели число, то увеличиваем count на единицу. Считываем строку из консоли и никуда ее не сохраняем. Выводим посчитанную сумму на экран. Если в строке есть несколько слов, разделенных пробелами, и первое из них — число, метод hasNextInt() вернет true, даже если остальные слова будут не числами. Поэтому наша программа будет правильно работать, только если в каждой строке написано не больше одного «слова».

```
Scanner console = new Scanner(System.in);  
int count = 0;  
for (int i = 0; i < 10; i++)  
{  
    if (console.hasNextInt())  
        count++;  
    console.nextLine();  
}  
System.out.println(count);
```

Давайте напишем программу, которая ничего не вводит, а скажем, что-нибудь вычисляет. Что-то сложное. Например, факториал числа 10. Факториалом числа n (обозначается n!) называется произведение ряда чисел: 1*2*3*4*5*...*n; В переменной f

будем хранить произведение чисел. Цикл от 1 до 10 (включительно). Умножаем f на очередное число (результат сохраняем в f). Выводим посчитанную сумму на экран. Стартовое значение f = 1, т.к. мы умножаем f на числа. Если бы f изначально было 0, произведение всех чисел на 0 дало бы 0.

```
int f = 1;
for (int i = 1; i <= 10; i++)
    f = f * i;
System.out.println(f);
```

Давайте напишем программу, которая рисует на экране треугольник: в первой строчке выводит 10 звездочек, во второй — 9 звездочек, затем 8, и т.д. Цикл по строкам (всего должно быть 10 строк). Вычисляем, сколько в строке должно быть звездочек. Цикл по звездочкам (выводим starCount звездочек). Добавляем перенос курсора на следующую строку, чтобы строки не слиплись. У нас тут должно быть два вложенных цикла — внутренний цикл должен выводить правильное количество звездочек в строке. А внешний цикл нужен для переключения по строкам.

```
for (int i = 0; i < 10; i++)
{
    int starCount = 10 - i;
    for (int j = 0; j < starCount; j++)
        System.out.print("*");
    System.out.println();
}
```

Давай используем цикл for, чтобы вывести на экран прямоугольный треугольник из восьмерок со сторонами (катетами) 10 и 10. То есть в первой строке должна быть одна 8, начиная слева, во второй - две и т.д.

```
public class Solution {
    public static void main(String[] args) {

        for (int i = 0; i < 10; i++)
        {
            int starCount = 1 + i;
            for (int j = 0; j < starCount; j++)
                System.out.print("8");
            System.out.println();
        }
    }
}
```

Обратите внимание, что ошибка округления вычисляется путем возведения в квадрат квадратного корня каждого числа. Затем результат вычитается из исходного числа, что и дает ошибку округления.

```
// Отображение квадратных корней чисел от 1 до 99 и ошибки округления.
class SqrRoot {
    public static void main(String[] args) {
        double num, sroot, rerr;
        for(num = 1.0; num < 100.0; num++) {
            sroot = Math.sqrt(num);
            System.out.println("Квадратный корень числа " + num +
                " равен " + sroot);
        }
    }
}
```



```
// Вычислить ошибку округления.
(sroot * sroot);
System.out.println("Ошибка округления составляет " + rerr);
System.out.println();
rerr = num
}}}
```

Например, следующая программа выводит числа от 100 до -95 с уменьшением значения переменной управления циклом на 5:

```
// Цикл for, выполняющийся в отрицательном направлении,
class DecrFor {
public static void main(String[] args) {
int x;
for(x = 100; x > -100; x -= 5) //На каждой итерации значение переменной управления
циклом уменьшается на 5
System.out.println(x);
}
}
```

```
// Использование запятых в операторе for.
class Comma {
public static void main(String[] args) {
int i, j;
for(i=0, j=10; i < j; i++, j--)
System.out.println("i и j: " + i + " " + j);
x += count;
Обратите внимание на
наличие двух переменных
управления циклом
}
}
```

Ниже показан вывод, генерируемый программой:

```
i и j: 0 10
i и j: 1 9
i и j: 2 8
i и j: 3 7
i и j: 4 6
```

В следующем примере цикл продолжает выполняться до тех пор, пока пользователь не введет с клавиатуры букву S:

```
// Цикл до тех пор, пока не будет введена буква S.
class ForTest {
public static void main(String[] args)
throws java.io.IOException {
int i;
System.out.println("Для остановки цикла нажмите S.");
for(i = 0; (char) System.in.read() != 'S'; i++)
System.out.println("Проход #" + i);
}
}
```

```
// Части цикла for могут быть пустыми.
class Empty {
public static void main(String[] args) {
int i;
for(i = 0; i < 10; ) {
System.out.println("Проход #" + i);
i++; // инкрементирование переменной управления циклом
Выражение итерации отсутствует
}}}
```

Инициализацию размещают вне цикла обычно лишь в ситуациях, когда для получения начального значения применяется сложный процесс, который не поддается заключению внутри оператора for.

```
// Вынесение за пределы определения цикла еще одной части,
class Empty2 {
public static void main(String[] args) {
int i;
i = 0; // вынести инициализацию за пределы цикла
for(; i < 10; ) {
System.out.println("Pass #" + i);
i++; // инкрементировать переменную управления циклом
}}}
```

Бесконечный цикл

```
for(;;) // намеренно бесконечный цикл
{
// ...
}
```

Циклы без тела

```
sum += i++
он эквивалентен такой последовательности операторов:
sum = sum + i;
i++;
```

```
// Тело цикла может быть пустым,
class Empty3 {
public static void main(String[] args) {
int i;
int sum = 0;
// Просуммировать числа от 1 до 5.
for(i = 1; i <= 5; sum += i++); //В этом цикле отсутствует тело!
System.out.println("Сумма равна " + sum);
}
```

Объявление переменных управления циклом внутри цикла for

```
class ForVar {
public static void main(String[] args) {
int sum = 0;
int fact = 1;
// Вычислить факториал чисел от 1 до 5.
for(int i = 1; i <= 5; i++) { //Переменная i объявлена внутри оператора for
sum += i; // переменная i известна во всем цикле
}
```

```

fact *= i;
}
// Но уже здесь переменная i не известна.
System.out.println("Сумма равна " + sum);
System.out.println("Факториал равен " + fact);
}
}

```

Д/з11 1) С клавиатуры вводится целое число, программа должна определить, является ли это число палиндромом. Например 12321, 77;

```

import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner kb= new Scanner(System.in);
        System.out.print("Введите число: ");
        int num= kb.nextInt();
        int reverseNum=0, initialNum, remainder;
        initialNum = num; //создаем новую переменную и сохраняем ввод.
        while(num!=0) ///Пока num не станет равным нулю, находим остаток от num и
        сохраняем его в переменной reverseNum
        {
            remainder= num % 10;
            reverseNum = (reverseNum * 10) + remainder;
            num = num / 10;
        }
        if (initialNum == reverseNum) //Определяем, равно ли initialNum числу
        reverseNum.
        {
            System.out.println(initialNum + " - это палиндром"); //Если оба равны, выводим,
            что это палиндром.
        }
        else
        {
            System.out.println(initialNum + " - это не палиндром."); // или это не палиндром.
        }
    }
}

```

2) С клавиатуры вводится номер автобусного билета, программа должна определить является ли билет счастливым. Счастливым называется билет, у которого сумма трех старших разрядов == сумме трех младших разрядов;

```

import java.util.Scanner;
class Main {
    public static void main(String[] args) {
        Scanner kb= new Scanner(System.in);
        System.out.print("Введите 6 цифр с билета: ");
        int number = kb.nextInt();
        int step = 1;
        int count = 0;
        int sum = 0;
        while (number != 0) {
            sum += step * (number % 10);

```

```

        number /= 10;
        if (++count == 3) {
            step = -step;
        }
    }
    if (count == 6 && sum == 0) {
        System.out.println("Это счастливый билетик");
    } else {
        System.out.println("Это не счастливый билетик");
    }
}
}
}

```

3) С клавиатуры вводится целое число и программа вычисляет факториал этого числа. Факториал - это произведение ряда чисел от 1 до указанного. Например $5! = 1*2*3*4*5 = ???$;

```

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите число: ");

        int num = kb.nextInt();
        int fact = 1;

        for (int i = 2; i <= num; i++) {
            fact = fact * i;
        }
        System.out.println(fact);
    }
}

```

4) С клавиатуры вводится основание и показатель степени, и программа вычисляет степень;

```

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите основание: ");
        int number = kb.nextInt();
        System.out.print("Введите показатель степени: ");
        int degree = kb.nextInt();

        System.out.println("Степень числа: " + Math.pow(number, degree));
    }
}

```

- 5) Вывести на экран ряд фибоначчи до указанного предела. Предел вводится с клавиатуры;

```
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите предел числа Фибоначчи: ");
        int n;
        n = kb.nextInt();

        int num1 = 0;
        int num2 = 1;

        for (int i = 1; i <= n; ++i) {
            System.out.println(num1 + " ");
            int sumOfPrevTwo = num1 + num2;
            num1 = num2;
            num2 = sumOfPrevTwo;
        }
    }
}
```

- 6) Вывести на экран заданное количество членов из ряда фибоначчи. Каждый следующий член ряда это сумма двух предыдущих. 0 1 (0+1)1 (1+1)2 (2+1)3

```
class Main {
    public static void main(String[] args) {
        int n;
        int num1 = 0;
        int num2 = 1;
        for(int i = 1; i <= 50; i++){
            System.out.println(num1 + " ");
            int sumOfPrevTwo = num1 + num2;
            num1 = num2;
            num2 = sumOfPrevTwo;
        }
    }
}
```

- 7) Вывести на экран таблицу умножения; $2 \times 2 = 4$ с клавиатуры ничего не вводится

```
class Main {
    public static void main(String[] args) {
        for (int a = 1; a <= 10; a++) {
            for (int b = 1; b <= 10; b++) {
                System.out.println(a + " x " + b + " = " + a * b);
            }
            System.out.println();
        }
    }
}
```

8) вывести на экран таблицу Пифагора; с клавиатуры ничего не вводится

```
class Main {
    public static void main(String[] args) {
        for (int a = 1; a < 10; a++) {
            for (int b = 1; b < 10; b++)
                System.out.printf("%3d", a * b);
            System.out.println();
        }
    }
}
```

9) Вывести на экран ряд простых чисел до указанного предела. Простым называется число, которое делится без остатка только на себя и на 1.

```
import java.util.Scanner;
class Main {
    public static void prime_N(int N) {
        int x, y, z;
        for (x = 1; x <= N; x++) {
            if (x == 1 || x == 0)
                continue;
            z = 1;
            for (y = 2; y <= x / 2; ++y) {
                if (x % y == 0) {
                    z = 0;
                    break;
                }
            }
            if (z == 1)
                System.out.print(x + " ");
        }
    }
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите лимит: ");
        int N = kb.nextInt();
        prime_N(N);
    }
}
```

тут выведены просто числа:

```
Scanner kb = new Scanner(System.in);
    System.out.print("Введите лимит: ");
    int N = kb.nextInt();
    for (int i = 0; i < n; i++)
    {
        if (i%10==0) System.out.println();
        System.out.print("%5d\t", i);
    }
    System.out.println();
```

а тут уже на 1 и на себя:

```

Scanner kb = new Scanner(System.in);
    System.out.print("Введите лимит: ");
    int N =kb.nextInt();
    for (int i = 0;i < n; i++)
    {
        Boolean simple = true; // предположим что число простое
        //но это нужно проверить
        for (int j = 2; j < i ; ji++) {

            if (i%j==0)
            {
                simple=false;
                break;
            }
        }
        if (i%10==0) System.out.println();
        System.out.print("%5d\t", i);
    }
    System.out.println();
}
}

```

В циклах применимы ключевые слова break и continue. break прерывает итерацию и все последующие, continue прерывает текущую итерацию и переходит к следующей.

```

Например, давайте вычислим сумму чисел от 1 до 100 и выведем ее
int sum = 0;
int num = 0;
while(num <= 100) {
    sum += num;
    num++;
}
System.out.println(sum); //Обратите внимание, что последний оператор print
находится вне области while.

```

Arithmetical overflow (Арифметическое переполнение) - при сложении или умножении больших целых чисел результат может не помещаться в переменную. Поскольку требует больше битов чем хранит эта переменная, такая ситуация называется арифметическим переполнением. Когда старший бит результата теряется и мы получаем неправильный результат. Данную ситуацию практически невозможно отследить, но ее можно предотвратить иногда использованием больших типов, а иногда ограничением операндов. Если есть необходимость вычисления очень больших целых чисел, то как правило используют длинную арифметику, где число хранится не в отдельной переменной, а в структуре данных и ограничивается это число лишь объемом оперативной памяти компьютера.

```

import java.util.Scanner;

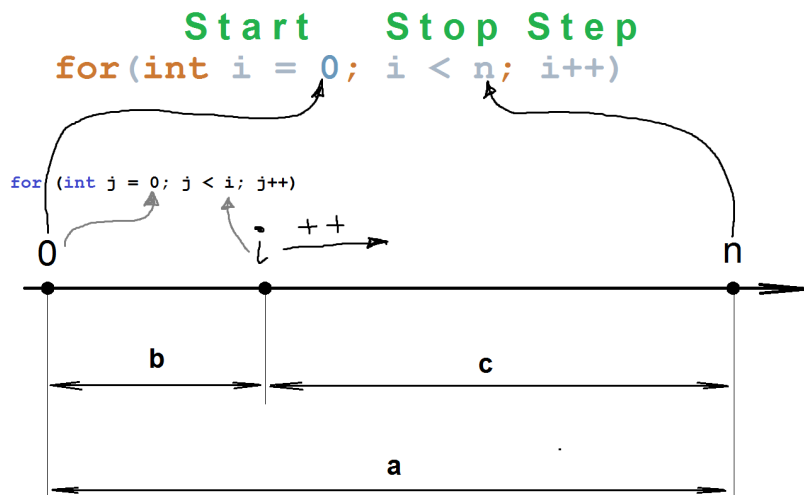
class Main {
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите размер фигуры: ");
        int n = kb.nextInt();
    }
}

```

```

for (int i=0; i<n; i++) //этот цикл повторяет вывод строки на экран
{
    for (int j=0; j<n; j++)
    {
        System.out.print(" "); // этот цикл повторяет вывод звездочки на экран
    }
    System.out.print(); //после вывод строки нужно перейти на новую строку
}
}
}

```



Д/з12 1) Применить длинную арифметику при решении задачи факториал и фибоначчи

```

import java.util.Scanner;
import java.math.BigInteger;

public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.println("Применение длинной арифметики при решении задачи
Factorial");
        System.out.print("Введите число: ");
        int num = kb.nextInt();
        BigInteger fact = BigInteger.valueOf(1);

        for (int i = 1; i <= num; i++) {
            fact = fact.multiply(BigInteger.valueOf(i));
        }
        System.out.println(fact);

        System.out.println("Применение длинной арифметики при решении задачи
Fibonacci");
        System.out.print("Введите предел числа Фибоначчи: ");
        int n = kb.nextInt();

        BigInteger num1 = new BigInteger("1");
        BigInteger num2 = new BigInteger("1");

        for (int i = 2; i <= n; ++i) {

```



```
        BigInteger c = num1;
        num1 = num2;
        num2 = c.add(num1);
    }
    System.out.println(num2);
}
}
```

2) Вывести треугольники по вводу запроса: использовать for
Вывести на экран следующие геометрические фигуры:

```

*
* *
* * *
* * * *
* * * * *

* * * * *
* * * *
* * *
* *
*

    * * * * *
      * * * *
        * * *
          * *
            *

              *
            * *
          * * *
        * * * *
      * * * * *

          *
        * *
      * * *
    * * * *
  * * * * * *
    * * * *
      * *
        *

* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

```
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите размер фигуры: ");
        int h = kb.nextInt();

        System.out.println("");
        System.out.println("1");

        for (int i = 0; i < h; i++) {
```

```

        for (int j = 0; j <= i; j++) {
            System.out.print("*");
        }
        System.out.println();
    }

    System.out.println();
    System.out.println("2");

    for (int i = h; i >= 0; i--){
        for (int j = 0; j < i; j++){
            System.out.print("*");
        }
        for (int j = h; j > i; j--){
            System.out.print(" ");
        }
        System.out.println();
    }

    System.out.println();
    System.out.println("3");

    for (int i = h; i >= 0; i--){
        for (int j = h; j > i; j--){
            System.out.print(" ");
        }
        for (int j1 = 0; j1 <= i; j1++){
            System.out.print("*");
        }
        System.out.println();
    }

    System.out.println();
    System.out.println("4");

    for (int i = 0; i < h; i++){
        for (int j = h; j > i; j--){
            System.out.print(" ");
        }
        for (int j1 = 0; j1 <= i; j1++){
            System.out.print("*");
        }
        System.out.println(" ");
    }

    System.out.println();
    System.out.println("5");

    for (int i = 0; i < h; i++){
        for (int j = h; j > i; j--){
            System.out.print(" ");
        }
        for (int j1 = 0; j1 <= i; j1++){
            System.out.print("*");
        }
    }

```

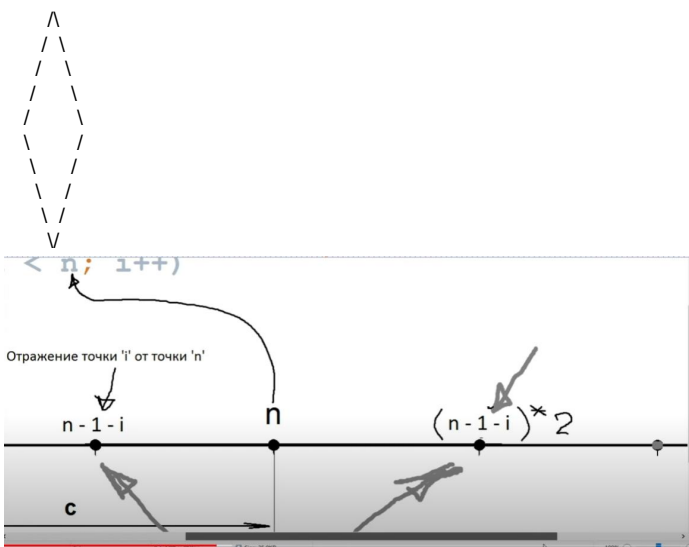
```

    }
    for (int j = 0; j < i; j++){
        System.out.print("*");
    }
    for (int j = h; j > i; j--){
        System.out.print(" ");
    }
    System.out.println(" ");
}
for (int i = h; i >= 0; i--){
    for (int j = h; j > i; j--){
        System.out.print(" ");
    }
    for (int j1 = 0; j1 <= i; j1++){
        System.out.print("*");
    }
    for (int j = 0; j < i; j++){
        System.out.print("*");
    }
    for (int j = h; j > i; j--){
        System.out.print(" ");
    }
    System.out.println();
}

System.out.println();
System.out.println("6");

for (int i = 0; i < h; i++) {
    for (int j = 0; j < h; j++) {
        System.out.print("* ");
    }
    System.out.println();
}
}
}

```



```

import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите размер фигуры: ");
        int n = kb.nextInt();

        for (int i=0; i<n; i++) // левая часть
        {
            for (int j = i; j < n; j++) System.out.print(" "); //выводит пробел
            System.out.print("/"); //выводим палку
            for (int j = 0; j < i*2; j++) System.out.print(" "); //правая верхняя часть
            System.out.print("\n");
            System.out.println();
        }

        for (int i=0; i<n; i++) // рисуем нижнюю часть увеличение на отрезке b 0
        {
            for (int j=0; j<=i; j++) System.out.print(" "); //увеличили количество итераций на
1, благодаря этому появляется лишний пробел
            System.out.print("\n");
            for(int j=0; j < (n-1-i)*2; j++) System.out.print(" "); // количество в каждой строке
увеличивается на два, for будет на с, он уменьшается. начинается в точке i и
заканчивается в точке n направо
            System.out.print("/"); //количество итерации надо уменьшить
            System.out.println(); // можно поставить n-1 и второй вариант i+1
        }
    }
}

```

Из квадратика можно вывести любую фигуру

```

+ - + - +
- + - + -
+ - + - +
- + - + -
+ - + - +

```

```

import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите размер фигуры: ");
        int n = kb.nextInt();

        for (int i=0; i<n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if ((i+j)%2 ==0) System.out.print("+ ");
                else System.out.print("- "); //по звездочкам видим куда едет курсор
                //или можно так System.out.print((i+j)%2 == 0 ? "+ " : "- ");
            }
        }
    }
}

```

```

    }
    System.out.println();
  }
}

```

Д/313

1)

```

*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****

      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****

*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****

      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****

*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****
*****      *****      *****

```

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Введите размерность доски: ");
        int n = sc.nextInt();

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    for (int l = 0; l < n; l++) {
                        if ((i + k) % 2 == 0)
                            System.out.print("* ");
                        else
                            System.out.print(" ");
                        /*тернарный метод System.out.print((i%2==k%2) ? "*" : " "); */
                    }
                }
            }
            System.out.println();
        }
    }
}
```

2. Вывести на экран треугольник Паскаля.

```
import java.util.Scanner;
```

```

public static void main(String[] args) {
    Scanner kb = new Scanner(System.in);
    System.out.print("Введите количество строк: ");
    int depth = kb.nextInt();
    int nf = 1; //factorial
    for (int i=0; i <= depth+1; i++) System.out.print(" "); // зеркалим то что в цикле ниже
    System.out.println(1); // 1 строка треугольника
    for (int n = 1; n <= depth; n++)
    {
        for (int j=0; j<depth-n; j++)
        {
            System.out.print(" ");
        }
        System.out.print(1 + " ");
        nf *= n;
        int mf = 1;
        for (int m = 1; m <= n; m++)
        {
            mf *= m;
            int kf = 1;
            for (int k = 1; k <= n-m; k++)
            {
                kf *= k;
            }
            int member = nf/mf/kf;
            System.out.print(member+ " ");
        }
        System.out.println();
    }
}

```

или так

```

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите количество строк: ");
        int n = kb.nextInt();

        for (int i = 0; i < n; i++) {
            int number = 1;
            System.out.format("%" + (n - i) * 2 + "s", "");
            for (int j = 0; j <= i; j++) {
                System.out.printf("%4d", number);
                number = number * (i - j) / (j + 1);
            }
            System.out.println("");
        }
    }
}

```

3. Бонус

https://github.com/okovtun/BV_011/blob/master/ControlStructures/Geometria/Task/ChessBoard.jpg

GK__Java_urok_04

Цикл for-each - для реализации этого цикла используется специальный синтаксис оператора for

```
for (тип элемент: коллекция) {  
    операция;  
}
```

Пример для массива

```
int [] numbers = new int[] { 3, 2, 1 };  
for (int number : numbers)  
{  
    System.out.print(number + " ");  
}
```

Пример для коллекции:

```
List<Integer> lists = new ArrayList<Integer>();  
lists.add(1);  
lists.add(3);  
for (  
    Integer  
value: lists)  
{  
    System.out.println(value); }
```

Цикл создает переменную, которая автоматически присваивается каждому значению массива во время цикла. Вы можете называть переменную как угодно: в нашем примере мы назвали ее x. Обратите внимание на двоеточие после переменной - оно читается как "для каждого x в числах".

```
int[] nums = {2, 3, 5, 7};  
for (int x: nums) {  
    System.out.println(x);  
}
```

Вычисления суммы всех значений массива:

```
int[] ages = {18, 33, 24, 64, 45};  
int sum = 0;  
for(int x: ages) {  
    sum += x;  
}  
System.out.println(sum);
```

Результат будет 8 . x=3, x> res true, потому что res=0. Далее res=3, x=8, x>res true. res=8,x=5, x>res false.

```
int[ ] nums = {3, 8, 5, 2};  
int res = 0;  
for(int x: nums) {
```

```
        if(x > res) {
            res = x;
        }
    }
    System.out.println(res);
```

Оператор break прерывает выполнение цикла и передает управление в конец тела цикла. Оператор break может быть использован совместно с меткой, например для одновременного прерывания нескольких вложенных циклов. В этом случае управление будет передано в конец блока помеченного меткой указанной после break.

Метка – это именует блок кода.

Программа будет считывать строки с клавиатуры, пока не будет введена строка exit.

```
Scanner console = new Scanner(System.in);
while (true)
{
    String s = console.nextLine();
    if (s.equals("exit"))
        break;
}
```

Вводим данные с клавиатуры и сразу их выводим на экран, пока не будет введено слово "enough". Слово "enough" выводить не нужно. Для этого необходимо использовать бесконечный цикл (while(true)).

```
public class Solution {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        while (true)
        {
            String text = scan.nextLine();
            if (text.equals("enough"))
                break;
            System.out.println(text);
        }
    }
}
```

```
two: for (int n = 0; n < 10; n++) {
    for (int j = 10; j > 0; j--) {
        System.out.print(j - n +
            "
");
        if (j + n == 5 && n > 0)
            break two;
    }
}
```

```
// Использование break для выхода из цикла,
class BreakDemo {
```



```

public static void main(String[] args) {
    int num;
    num = 100;
    // Цикл до тех пор, пока квадрат i меньше num.
    for(int i=0; i < num; i++) {
        if(i*i >= num) break; // прекратить выполнение цикла
        if i*i >= 100 System.out.print{i + " ");
    }
    System.out.println("Цикл завершен.");
}
}

```

При использовании внутри набора вложенных циклов оператор break производит выход только из самого внутреннего цикла

```

// Использование break с вложенными циклами,
class Break3 {
    public static void main(String[] args) {
        for(int i=0; i<3; i++) {
            System.out.println("Счетчик внешнего цикла: " + i);
            System.out.print(" Счетчик внутреннего цикла: ");
            int t = 0;
            while(t < 100) {
                if(t == 10) break; // прекращает выполнение цикла, если t равно 10
                System.out.print(t + " ");
                t++;
            }
            System.out.println();
        }
        System.out.println("Циклы завершены .");
    }
}

```

С оператором break связаны еще два момента, о которых следует помнить. Во-первых, в цикле могут находиться более одного оператора break. Однако будьте осторожны. Слишком большое количество операторов break способно деструктурировать код. Во-вторых, оператор break, завершающий switch, влияет только на этот оператор switch, но не на любые объемлющие циклы.

Общая форма оператора break с меткой выглядит следующим образом:

break метка;

В показанной далее программе реализованы три вложенных блока:

```

// Использование оператора break с меткой.
class Break4 {
    public static void main(String[] args) {
        int i;
        for(i=1; i<4; i++) {
            one: {
                two:
                three:
                {

```

```

{
System.out.println("\ni равно " + i);
if(i==1) break one; ///Перейти по метке
if(i==2) break two;
if(i==3) break three;
// Эта строка код никогда не будет достигнута.
System.out.println("не выводится");
}
System.out.println("После блока three.");
}
System.out.println("После блока two.");
}
System.out.println("После блока one.");
}
System.out.println("После цикла for.");
}
}
}

```

Оператор break применяется для выхода из нескольких вложенных циклов for. Когда во внутреннем цикле выполняется оператор break, управление переходит в конец блока, определенного внешним циклом for, который помечен посредством done. В итоге оставшаяся часть всех трех циклов будет пропущена.

```

// Еще один пример использования оператора break с меткой,
class Break5 {
public static void main(String[] args) {
done:
for(int i=0; i<10; i++) {
for(int j=0; j<10; j++) {
for(int k=0; k<10; k++) {
System.out.println(k + " ");
if(k == 5) break done; // переход по метке done
}
System.out.println("После цикла k"); // не выполнится
}
System.out.println("После цикла j"); // не выполнится
}
System.out.println("После цикла i");
}
}

```

```

// Метка находится перед оператором for.
stop1: for(x=0; x < 5; x++) {
for(y = 0; y < 5; y++) {
if(y == 2) break stop1;
System.out.println("x и y: " + x + " " + y);
}
}
System.out.println();
// Метка находится непосредственно перед {.
for(x=0; x < 5; x++)
stop2: {
for(y = 0; y < 5; y++) {
if(y == 2) break stop2;
}
}
}

```

```
System.out.println("x и y: " + x + " " + y);
}}}
```

// Эта программа содержит ошибку, Поскольку цикл , помеченный как one, не охватывает оператор break, передать управление из этого блока невозможно.

```
class BreakErr {
public static void main(String[] args) {
one: for(int i=0; i<3; i++) {
System.out.print("Проход " + i + ": ");
}
for(int j=0; j<100; j++) {
if(j == 10) break one; // ОШИБКА
System.out.print(j + " ");
}}}
```

Оператор continue прерывает выполнения тела цикла и передает управление в начало цикла. Происходит пропуск текущей итерации цикла.

```
int i = 0;
while (i < 10)
{
i++;
if (i % 2 == 0)
continue;
System.out.println(i);
}
```

Оператор continue также может применяться с меткой

```
one: for (i = 0; i < 10; i++) {
for (int j = 0; j < 10; j++) {
if (i == j) continue one;
System.out.printf("i=%d j=%d \n", i, j);
}
}
```

Задача: мы хотим написать программу, которая выводит на экран 20 чисел от 1 до 20, и при этом пропускает числа, которые делятся на 7. Вот как мог бы выглядеть этот код: На самом деле такой код работать не будет, потому что *i* навсегда застынет на цифре 7. Ведь вызов `continue` привел к тому, что пропустили две команды — `System.out.println(i)` и `i++`. Поэтому достигнув значения 7, переменная *i* перестанет меняться, и цикл будет выполняться вечно. Мы специально написали такой код, т.к. это очень распространенная ошибка. Как же его исправить?

Тут есть два варианта: Вариант 1: изменить *i* перед вызовом `continue`, но после вызова `i % 7`

Вариант 2: увеличивать *i* всегда в начале цикла. Но тогда стартовое значение *i* должно быть 0.

```
int i = 1;
while (i <= 20)
{
```

```
    if ( ( i % 7) == 0 )
    {
        i++;
        continue;
    }

    System.out.println(i);
    i++;
}
```

```
int i = 0;
while (i < 20)
{
    i++;
    if ( ( i % 7) == 0)
        continue;
    System.out.println(i);
}
```

Вывести на экран сумму чисел от 1 до 100 включительно, не кратных 3. Для этого используй цикл while. Подсказка: чтобы в цикле перейти к следующему числу, используй оператор continue.

```
public class Solution {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int sum = 0;
        int i = 0;
        while (i++ < 100)
        {

            if ((i % 3) == 0)
                continue;
            sum = sum + i;
        }
        System.out.println(sum);
    }
}
```

```
// Использование оператора continue,
class ContDemo {
public static void main(String[] args) {
int i;
// Вывести четные числа между 0 и 100.
for(i = 0; i<=100; i++) {
if((i%2) != 0) continue;
System.out.println(i);
// следующая итерация
}
}
}
```

Оператор continue в циклах while и do-while обеспечивает передачу управления непосредственно условному выражению и продолжение циклического процесса. В случае for вычисляется выражение итерации цикла, затем вычисляется условное выражение, после чего цикл продолжается. Как и в случае break, в операторе continue может присутствовать метка, которая указывает, какой охватывающий цикл необходимо продолжить. Далее приведен пример программы, где используется оператор continue с меткой:

```
// Использование оператора continue с меткой,
class ContToLabel {
public static void main(String[] args) {
outerloop:
for(int i=1; i < 10; i++) {
System.out.print("XnПроход внешнего цикла #" + i + ", внутренний цикл: ");
for(int j = 1; j < 10; j++) {
if(j == 5) continue outerloop; // продолжить внешний цикл
System.out.print(j);
}}}}
}
```

```
/* Упражнение 3.3. Законченная справочная системы по операторам Java,
позволяющая обрабатывать множество запросов. */
class He1p3 {
public static void main(String[] args)
throws java.io.IOException {
char choice, ignore;
for (;;) {
do {
System.out.println("Справка no:");
System.out.println("
System.out.println("
System.out.println("
System.out.println("
System.out.println("
System.out.println("
System.out.println("
System.out.print("Выберите вариант (или q для завершения): ");
choice = (char) System.in.read();
do {
1. i f " ) ;
2. switch");
3. for");
4. while");
5. do-while");
6. break");
7. continueXn");
ignore = (char) System.in.read();
} while(ignore != 'Xn 1');
} while( choice < '1'|choice > '7' & choice != 'q');
if(choice == 'q') break;
System.out.println("\n");
switch(choice) {
case '1':
System.out.println("Оператор if:\n");
}
```

```

System.out.println("if(условие) оператор;");
System.out.println("else оператор;");
break;
case '2':
System.out.println("Традиционный оператор switch:\n");
System.out.println("switch(выражение) {");
System.out.println("
System.out.println("
System.out.println("
System.out.println(" // ...");
System.out.println("}");
break;
case '3':
System.out.println("Цикл for:\n");
System.out.print("for(инициализация; условие; итерация)");
System.out.println(" оператор;");
break;
case 4':
System.out.println("Цикл while:\n");
System.out.println("while(условие) оператор;");
break;
case '5':
System.out.println("Цикл do-while:\n");
System.out.println("do {");
System.out.println(" оператор;");
System.out.println("} while (условие);");
break;
case '6':
System.out.println("Оператор break:\n");
System.out.println("break; или break метка;");
break;
case *7':
System.out.println("Оператор continue:\n");
System.out.println("continue; или continue метка;");
break;
}
System.out.println();
}}}

```

Оператор return прекращает выполнение метода, возвращая управление в место вызова метода.

Основные положения Java Code Convention

Классы должны именоваться с большой буквы, английскими символами без пробелов. Имя класса должно соответствовать одному объекту этого класса.

```

class Cats // плохо
class Cat // хорошо

```

Интерфейсы должны именоваться по правилам классов. Не желательно использование дополнительных указаний в имени интерфейса на принадлежность к интерфейсу.

```

interface ICloneable // плохо
interface Cloneable // хорошо

```

Если в имени класса содержится аббревиатура содержащая больше двух букв, все буквы аббревиатуры кроме первой должны быть строчными.

```
class FPSRenderer // плохо
class FpsRenderer // хорошо
```

Объявление (декларация) нескольких переменных должно быть в разных строках.

```
int a, b; // плохо
int a; // хорошо
int b; // хорошо
```

Если возможно инициализировать переменные в месте объявления.

```
int a;
...
a = 255; // плохо
int a = 255; // хорошо
```

Необходимо объявлять переменные как можно ближе к месту использования.

```
int a = 255; // плохо
System.out.println("file")
a + = 127;
System.out.println("file")
int a = 255; // хорошо
int c = a + 127;
```

При объявлении массива необходимо указывать скобки после указания типа переменной, а не после имени.

```
int a [] = new int[3]; // плохо
int[] a = new int[3]; // хорошо
int [] a [] = new int[3]; // плохо
int[][] a = new int[3][3]; // хорошо
```

Константы принято именовать большими буквами, а в качестве разделителя использовать символ подчеркивания.

```
final int MAX_STEP = 3; // хорошо
final String DELIMITER = «;»;// хорошо
```

Методы должны именоваться латинскими буквами, начиная с маленькой буквы. Первой слово в названии метода должно быть глаголом или наречием. Если в имени метода содержится более одного слова, каждое слово кроме первого выделяется большой буквой (Camel нотация).

```
void startprocess() {} // плохо
void Start() {} // плохо
void start_process() {} // плохо
void sProcess() {} // плохо
void sp() {} // плохо
void start() {} // хорошо
void startProcess() {} // хорошо
```

Системы контроля версий (22.08.2023)

Система контроля версий (VCS) version control system - это программа которая сохраняет промежуточные контрольные точки кода и позволяет любой момент откатываться на эти контрольные точки. существует множество vsc такие table version мерсигу, но коронное место занимает GIT потому что он позволяет использовать

платформу GitHub. Git != GitHub. **Git** - это система контроля версий, а GitHub - это облачное хранилище в котором используется VCS Git.

Ключевым понятием контроля версий является репозиторий. **Репозиторий** - это хранилище исходных кодов. Кроме исходных кодов репозиторий так же можно хранить простые текстовые файлы. Файлы изображение, звуковые и видео файлы и вообще все что угодно. Потому что репозиторий - это самая обычная папка с файлами и в ней может быть все что угодно.

Существует понятие локального и удаленного репозитория. Локальный репозиторий находится на локальном компьютере программиста. А удаленный находится на корпоративном сервере или на GitHub. Локальный и удаленный репозиторий нужно постоянно синхронизировать, для этого используются команды push (отправить все изменения с локального репозитория на удаленный), pull (скачать и распаковать изменения с удаленного репозитория на локальный репозиторий) и fetch (скачать, но не распаковывать изменения с удаленного репозитория на локальный репозиторий).

Следующим ключевым понятием контроля версий является commit. **Commit** - это контрольная точка, в которой сохраняются исходные коды программы, и на эту контрольную точку всегда можно откатить состояние репозитория или отдельного файла. На удаленный репозиторий можно отправить только контрольную точку. Если на удаленный репозиторий нужно отправить отдельный файл то это файл добавляется в commit и на удаленный репозиторий отправляется commit. Для того чтобы управлять версиями приложения нужно установить систему контроля версий.

Команды GitHub <https://git-scm.com/docs/git-rm>
<https://git-scm.com/book/en/v2>
https://www.w3schools.com/git/git_ignore.asp?remote=github

Не все файлы репозитория являются критически важными. Очень часто компилятор создает некоторые служебные файлы в процессе сборки приложения. Эти файлы создаются при каждой сборке и нет необходимости хранить их в репозитории и тем более заливать на удаленный репозиторий такие файлы и папки перечислены в папке .gitignore Большинство ide автоматически формируют это файл. Но при необходимости этот файл можно сформировать вручную в текстовом редакторе.

1. Правой кнопкой мыши по нужной папке и выбираем Open Git Bash here
2. **git status** ↵ проверить статус папок/ показывает состояние репозитория
3. **git init** ↵ создаем локальный папку/ git-репозиторий в текущем каталоге
4. **git add *** ↵ добавить в локальный репозиторий все файлы в staged-area. Все файлы в Staged-area войдут в следующий commit;
5. **git add .*** ↵ добавить скрытый файл в локальный репозиторий. добавляются отдельной командой
6. **git commit -m 'init'** ↵ -m означает message, где 'init' можно писать все что угодно
7. **git config --global user.email "olesyashka@yandex.ru"** ↵
8. "Ctrl + W"
"Alt + Backspace" удаляет то, что писали до точки
будет git config --global user.
9. **git config --global user.name "Lesya"** ↵
10. **git commit -m 'init'** ↵
11. **git log** ↵ это журнал отображает историю локального репозитория
все изменения хранятся в папке .git
12. на <https://github.com/> создаем пустой репозиторий
+ new repository - Repository name - нап lessons - выбираем public - create repository
...or create a new repository on the command line
echo "# lessons" >> README.md


```

git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/OlesyaVolkova/DZ18.git
git push -u origin main
...or push an existing repository from the command line
git remote add origin https://github.com/OlesyaVolkova/DZ18.git
git branch -M main
git push -u origin main
...or import code from another repository
You can initialize this repository with code from a Subversion, Mercurial, or TFS
project
13. git remote add origin https://github.com/OlesyaVolkova/DZ18.git ↵
    Указываем путь откуда с удаленного репозитория и на какой сервер на github
    залить локальный репозиторий.
14. git remote -v ↵ проверить добавился ли локальный репозиторий. push с
    локального на удаленный репозиторий
15. git push origin master Залить commit на github. Есть master -> master все ok
    Отправляет локальные коммиты на удаленный репозиторий
16. git remote remove origin отвязывает локальный репозиторий от удаленного
17. Ctrl + D выход из Open Git Bash here

1. Правой кнопкой мыши по нужной папке и выбираем Open Git Bash here
2. git clone https://github.com/OlesyaVolkova/lessons.git ↵ скачать с github

```

Д/з14 Залить на GitHub 5 проектов на выбор

Массивы (29.08.2023)

Массив (Array) - это множество переменных одного типа в непрерывной области памяти. В языке Java массивы также являются объектами, т.е. они описаны при помощи классов. Отличием массивов от примитивных типов является то, что массив это ссылочный тип данных. Т.е. объект Array не хранит сами данные, а лишь хранит ссылку на них. **Ссылка (Reference)** - это переменная которая содержит адрес другой переменной. Сам же массив хранится в Heap. **Heap** - это динамическая память. Все же значащие типы такие как `int`, `char`, `double` хранятся в так называемом **stack**. **Stack (стопка)** - статическая память, это модель памяти, из которой, последний записанный элемент считывается первым. При объявлении массива обязательно нужно указать его размер (количество элементов).

Массивы объявляются следующим образом:

```
type [ ] name = new type [size];
```

`type` - тип элементов массива. Все элементы массива могут быть только одного типа.
`[]` - показывают, что объявляется массив, а не одна переменная.
`name` - имя массива. Для именования массива используются такие же идентификаторы как и для именования переменных.
`new type [size];` - оператор `new` выделяет память для `size` элементов заданного типа (`type`) и возвращает адрес выделенной памяти (ссылку на выделенную память). Оператор `new` выделяет память из `heap`. Фактически он запрашивает эту память у JVM. Если у JVM имеется запрашиваемый объем памяти, то она ее выделяет, в противном случае возникает исключительная ситуация (`Exception`).

Ссылку на одну и ту же область динамической памяти могут содержать несколько объектов, в этом есть свои преимущества и недостатки. Такой подход используется с очень большой осторожностью. Потому что изменяя один объект, изменяются все объекты. После удаления последней ссылки на массив он удаляется из динамической памяти сборщиком мусора (Garbage collector).

Обращение к элементам массива

Для обращения к элементу массива нужно указать его номер в [] после имени массива. К элементам массива можно обращаться как на чтение так и на запись.

```
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите размер массива: ");
        int n = kb.nextInt();
        int [] arr = new int [n];
        arr [2] = 1024;
        System.out.println(arr [2]); //Обращение к элементу массива на чтение
    }
}
```

Для обращения к элементам массива очень удобно использовать цикл for потому что у него есть счетчик. Счетчик i который очень удобно использовать в качестве номера элемента.

```
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите размер массива: ");
        int n = kb.nextInt();
        int[] arr = new int[n];
        arr[2] = 1024;
        System.out.println(arr[2]);
        for (int i = 0; i < n; i++)
        {
            System.out.print(arr[i] + "\t");
        }
        System.out.println();
    }
}
```

Элементы массива всегда нумеруются с 0. Именно поэтому номер последнего элемента на 1 меньше количества элементов.



В массиве из пяти элементов отсутствует пятый элемент. К последнему элементу (на картинке 4) всегда можно обратиться так $n-1$.

Д/з15 В проекте 'Arrays' решить следующие задачи:

1. Ввести элементы массива с клавиатуры;
2. Вывести все элементы массива на экран от начала до конца;
3. Вывести все элементы массива на экран от конца к началу;
4. Вычислить сумму элементов массива;
5. Вычислить среднее-арифметическое элементов массива;
6. Найти минимальное и максимальное значение в массиве;

```
import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите размер массива: ");
        int n = kb.nextInt();
        int[] arr = new int[n];

        System.out.println("1. Ввести элементы массива с клавиатуры");
        System.out.print("Введите элементы массива: ");
        for (int i = 0; i < n; i++) {
            arr[i] = kb.nextInt();
        }
        System.out.print("Массив: ");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();

        System.out.println("2. Вывести все элементы массива на экран от начала до конца");
        int [] a = new int [] {3, 5, 8, 13, 21};
        for (int i=0; i< a.length; i++)
        {
            System.out.print(a[i] + " ");
        }
        System.out.println();

        System.out.println("3. Вывести все элементы массива на экран от конца к началу");
        int [] b = new int [] {3, 5, 8, 13, 21};
        for (int i=b.length-1; i>=0 ; i--)
        {
            System.out.print(b[i] + " ");
        }
        System.out.println();

        System.out.println("4. Вычислить сумму элементов массива");
        int[] c = {18, 33, 24, 64, 45};
        int sum = 0;

        for(int x=0;x<c.length;x++) {
```

```

        sum += c[x];
    }
    System.out.println(sum);

    System.out.println("5. Вычислить среднее-арифметическое элементов массива");
    int[] d = {10, 20, 30, 40, 50};
    int summa = 0;
    for(int x=0;x<d.length;x++) {
        summa += d[x];
    }
    System.out.println((double)summa/d.length);

    System.out.println("6. Найти минимальное и максимальное значение в массиве");
    int [] e = new int [] {36, 55, 78, 13, 201};
    int max= Integer.MIN_VALUE;
    int min= Integer.MAX_VALUE;

    for (int i = 0; i < e.length; i++) {
        max = Math.max(max, e[i]);
    }
    System.out.println("Max: " + max);

    for (int i = 0; i < e.length; i++) {
        min = Math.min(min, e[i]);
    }
    System.out.println("Min: " + min);    }}

```

так

```

import java.util.Scanner;

class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите размер массива: ");
        int n = kb.nextInt();
        int[] arr = new int[n];
        arr[2] = 1024; //обращение элементов массива на запись
        System.out.println(arr[2]);

        //Ввод элементов с клавиатуры
        System.out.println("Введите значение элементов");
        for (int i = 0; i < n; i++)
        {
            arr[i] = kb.nextInt();
        }
        //Вывод массива на в прямом порядке
        for(int i = 0; i < n; i++)
        {
            System.out.print(arr[i] + "\t");
        }
        System.out.println();

        //Вывод массива на экран в обратном порядке
    }
}

```

```

for (int i = n-1; i >= 0; i--)
{
    System.out.print(arr[i] + "\t");
}
System.out.println();

int sum =0;
for(int i=0; i<n; i++)
{
    sum +=arr[i];
}
System.out.println("Сумма элементов массива: " + sum);
System.out.println("Среднее арифметическое: " + (double)sum/ n);

//Поиск минимального и максимальное значение
int min, max;
// min = max = arr[0];
max= Integer.MIN_VALUE;
min= Integer.MAX_VALUE;

for (int i=0; i<n; i++)
{
    if(arr[i] < min) min = arr[i];
    if(arr[i] > max) max = arr[i];
}
System.out.println("Минимальное значение в массиве: " + min);
System.out.println("Максимальное значение в массиве: " + max);
}
}

```

или так

```

import java.util.Scanner;
import java.util.stream.IntStream;

class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите размер массива: ");
        int n = kb.nextInt();
        int[] arr = new int[n];

        System.out.println(arr[2]);

        //Ввод элементов с клавиатуры
        System.out.println("Введите значение элементов");
        for (int i = 0; i < n; i++)
        {
            arr[i] = kb.nextInt();
        }

        System.out.println("Сумма элементов массива " + IntStream.of(arr).sum());
        System.out.println("Средне арифметическое " + IntStream.of(arr).average());
        System.out.println("Минимальное значение в массиве " +
IntStream.of(arr).min().getAsInt());
    }
}

```

```
System.out.println("Максимальное значение в массиве " +  
    IntStream.of(arr).max().getAsInt());  
}  
}
```

Заполнение массива случайными числами (31.08.2023)

Для генерации случайных чисел используется класс **Random**. Для его использования его нужно импортировать `import java.util.Random;` и создать объект данного класса. Для того чтобы в каждый элемент массива положить случайное число надо обратиться к каждому элементу массива. Для генерации случайного числа определенного типа в классе `random` есть соответствующие методы.

Объект `random` всегда генерирует разные случайные числа. Для того чтобы `random` генерировал всегда одинаковые случайные числа при создании объекта нужно задать точку отсчета. 0 - seed (зерно, семена). `Random rand = new Random(0);` Для того чтобы задать верхний предел генерации случайных чисел нужно передать его как параметр методу `.nextInt()`; Например `arr[i]=rand.nextInt(200);` 200 - bound(связывать).

Если мы хотим задать нижний предел генерации, то `.nextInt` вызывается с двумя параметрами `origin & bound`. Нижний предел `Origin` можно задать только с верхним, его нельзя задать самостоятельно. `arr[i]=rand.nextInt(100, 200);`

Если мы задаем точку отсчета `seed`, то `random` начинает генерировать псевдослучайные числа.

```
import java.util.Random;  
import java.util.Scanner;  
  
class Main {  
    public static void main(String[] args) {  
        Scanner kb = new Scanner(System.in);  
        System.out.print("Введите размер массива: ");  
        int n = kb.nextInt();  
        int[] arr = new int[n];  
        arr[2] = 1024; //обращение элементов массива на запись  
        System.out.println(arr[2]);  
  
        //Ввод элементов с клавиатуры  
        System.out.println("Введите значение элементов");  
        for (int i = 0; i < n; i++)  
        {  
            arr[i] = kb.nextInt();  
        }  
  
        Random rand = new Random(0);  
        for(int i = 0; i < n; i++)  
        {  
            arr[i]=rand.nextInt(100, 200);  
        }  
  
        //Вывод массива в прямом порядке  
        for(int i = 0; i < n; i++)  
        {  
            System.out.print(arr[i] + " ");  
        }  
        System.out.println();  
    }  
}
```

```

//Вывод массива на экран в обратном порядке
for (int i = n-1; i >= 0; i--)
{
    System.out.print(arr[i] + "\t");
}
System.out.println();

int sum =0;
for(int i=0; i<n; i++)
{
    sum +=arr[i];
}
System.out.println("Сумма элементов массива: " + sum);
System.out.println("Среднее арифметическое: " + (double)sum/ n);

//Поиск минимального и максимальное значение
int min, max;
// min = max = arr[0];
max= Integer.MIN_VALUE;
min= Integer.MAX_VALUE;

for (int i=0; i<n; i++)
{
    if(arr[i] < min) min = arr[i];
    if(arr[i] > max) max = arr[i];
}
System.out.println("Минимальное значение в массиве: " + min);
System.out.println("Максимальное значение в массиве: " + max);
}
}

```

Д/з16 <https://github.com/OlesyaVolkova/DZ16.git>

- 1) Язык java. Написать код Есть массив из 10 элементов инициализированный последовательностью чисел от 0 до 10. Необходимо циклически сдвинуть этот массив на заданное число. элементов вправо. Количество сдвигов пользователь вводит с клавиатуры.

0 1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9 0 сдвинули на один элемент влево

2 3 4 5 6 7 8 9 0 1 сдвинули на один элемент влево

3 4 5 6 7 8 9 0 1 2 сдвинули на один элемент влево

итого циклически сдвинули на три элемента влево

- 2) Повторить предыдущее задание только сдвиг вправо.

- 3) Отсортировать массив в порядке возрастания.

<https://javarush.com/groups/posts/3840-kofe-breyk-136-sortirovka-massiva-v-porjadk-e-vozrastaniya-ili-ubihvaniya-s-pomojshjhju-arraysso> убывание и возрастание

тут 5 выбранный элемент, остальные это перебираемые элементы

5 7 8 3 4 1 0 | 7 не < 5 то не меняем места, 8 не < 5, если 3 < 5, то меняем местами:

первый проход

3 7 8 5 4 1 0 | 4 < 3, 1 < 3 - получается

1 7 8 5 4 3 0 | 0 < 1 - получается

0 7 8 5 4 3 1 |

второй проход:

0 7 8 5 4 3 1 | 7 - выбранный элемент. 8 < 7, 5 < 7

0 5 8 7 4 3 1 | 4 < 5

```
0 4 8 7 5 3 1 | 3 < 4
0 3 8 7 5 4 1 | 1 < 3
0 1 8 7 5 4 3 |
```

При втором заходе и далее 0-ой элемент не затрагиваем, потому что он уже отсортирован. При третьем заходе выбранный элемент будет 8, так как 0 и 1 уже отсортировали

```
import java.util.Scanner;
import java.util.Arrays;

class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.println("Массив: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]");
        System.out.print("Введите количество сдвигов: ");
        int k = kb.nextInt();
        int[] arr = new int[]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

        System.out.print("Сдвиг влево: ");
        for (int a = 0; a < k; a++) {
            int left = arr[0];
            for (int b = 0; b < arr.length - 1; b++)
                arr[b] = arr[b + 1];
            arr[arr.length - 1] = left;
        }
        System.out.println(Arrays.toString(arr));

        System.out.print("Сдвиг вправо: ");
        for (int i = 0; i < arr.length; i++) {
            arr[i] = i;
        }

        for (int i = 0; i < k; i++) {
            int right = arr[arr.length - 1];

            for (int j = arr.length - 1; j > 0; j--) {
                arr[j] = arr[j - 1];
            }
            arr[0] = right;
        }
        System.out.println(Arrays.toString(arr));
        System.out.println();

        System.out.println("Сортировка массива в порядке возрастания");
        System.out.println("Массив: [8, 9, 125, 2, 1, 80, 10]");
        int[] array = {8, 9, 125, 2, 1, 80, 10};
        Arrays.sort(array);
        System.out.println(Arrays.toString(array)); //Чтобы вызвать статический метод,
        следует указать перед ним имя класса Arrays
    }
}
```


Сортировка массива

```
import java.util.Scanner;
import java.util.Arrays;

class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите размер массива: ");
        int n = kb.nextInt();
        int[] arr = new int[n];

        for(int i = 0; i < n; i++) //счетчик i выбирает элемент в который надо поместить
минимальное значение
        {
            for(int j = i+1; j < n; j++) { //сравниваем со следующего за i, счетчик j перебирает
оставшиеся элементы
                // arr[i] выбранный элемент
                //arr [j] перебираемый элемент
                //если перебираемый элемент меньше чем выбранный, меняем их местами
                if(arr[j] < arr[i])
                {
                    int buffer = arr[i];
                    arr[i] = arr [j];
                }
            }
        }
        System.out.println(Arrays.toString(arr));
    }
}
```

Генерируются случайные уникальные числа

```
//UniqueRand
import java.util.Arrays;
import java.util.Random;

public class Main {
    public static void main(String[] args) {
        int n = 10;
        int[] arr = new int[n];
        Random rand = new Random(0);
        for(int i = 0; i < n; i++)
        {
            boolean unique; //флаг уникальности
            do {
                arr[i] = rand.nextInt(n); //верхний предел указываем как кол-во элементов
                unique = true; //предположим, что генерировалось уникальное случайное
число
                // но это нужно проверить:
                for(int j=0; j<i; j++)
                {
                    if(arr[i] == arr[j])
                    {
                        unique = false; //если раньше уже было такое число, то false
                    }
                }
            } while (!unique);
        }
    }
}
```

```

        break;
    }
}
}while (!unique);
}
System.out.println(Arrays.toString(arr));
}
}

```

или так

```

import java.util.Arrays;
import java.util.Random;

public class Main {
    public static void main(String[] args) {
        int n = 10;
        int[] arr = new int[n];
        Random rand = new Random(0);

        for(int i=0; i<n; i++)
        {
            arr[i]=rand.nextInt(n);
            for (int j=0; j<i; j++)
            {
                if (arr[i]==arr[j])
                {
                    i--;
                    break;
                }
            }
        }
        System.out.println(Arrays.toString(arr));
    }
}

```

Остаток от деления строго меньше делителя (05/09/2023)

$155_{10} = 10011011$ остатки пишем с последнего разряда от деления

Handwritten diagram illustrating the conversion of the decimal number 155 to binary using the division-by-2 method. The number 155 is written at the top left. Below it, a series of divisions by 2 are shown in a staircase pattern. The remainders (0 or 1) are circled and written to the left of each division step. The final binary result, 10011011, is written at the top right. The diagram is drawn on a grid background.

```

import java.util.Arrays;
import java.util.Scanner;

```

```

public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.println("Введите десятичное число: ");
        int decimal = kb.nextInt();
        int[] bin = new int[32]; //32 потому что 32 бита в Int
        int i = 0; //счетчик разрядов
        while (decimal > 0) {
            bin[i] = decimal % 2; //остаток от деления
            decimal /= 2; //decimal = decimal / 2
            i++; // тут сохраняется количество разрядов, они формируются с 0
        }
        // System.out.println(Arrays.toString(bin)); //тут вывод прямой, а не в обратном
        //порядке
        for (--i; i >= 0; i--) { //пишем код в обратном порядке, вначале --i максимальный
            //номер разряда на 1 меньше количества
            System.out.print(bin[i]);
        }
    }
}

```

или так

```

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.println("Введите десятичное число: ");
        int decimal = kb.nextInt();
        int[] bin = new int[32]; //32 потому что 32 бита в Int
        int i = 0; //счетчик разрядов
        while (decimal > 0) {
            bin[i++] = decimal % 2; //остаток от деления
            decimal /= 2; //decimal = decimal / 2
        }
        for (--i; i >= 0; i--) { //пишем код в обратном порядке
            System.out.print(bin[i]);
        }
    }
}

```

еще так

```

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.println("Введите десятичное число: ");
        int decimal = kb.nextInt();
        int[] bin = new int[32]; //32 потому что 32 бита в Int
        int i = 0; //счетчик разрядов
    }
}

```

```

    for (; decimal > 0; decimal /= 2)
        bin[i++] = decimal % 2;
    for (--i; i >= 0; i--) { //пишем код в обратном порядке
        System.out.print(bin[i]);
    }
}
}
}

```

Д/з17 <https://github.com/OlesyaVolkova/DZ17>

1) Есть массив из 10 элементов заполненный случайными числами (числа НЕ уникальные). Необходимо найти повторяющиеся значения и вывести их на экран и вывести на экран количество повторений каждого из этих значений.

```

import java.util.Arrays;
import java.util.Random;

public class Main {
    public static void main(String[] args) {
        int n = 10;
        int[] arr = new int[n];
        Random rand = new Random(0);

        for(int i=0; i<n; i++) //заполняем массив случайными числами
        {
            arr[i]=rand.nextInt(n/2);
        }
        System.out.println(Arrays.toString(arr));
        for (int i=0; i<n; i++) //текущее значение сравниваем как при сортировке
        {
            /* Прежде чем считать повторения нужно выяснить, встречался ли это элемент в
            массиве ранее. Если он там встречался ранее, то мы уже посчитали количество
            повторений для этого значения */
            boolean met_before=false;
            for(int j=0; j< i; j++) //просканировать левую часть массива
            {
                if (arr[i] == arr[j])
                {
                    met_before =true; //запомнить что было повторение
                    break;
                }
            }
            /*если элемент встречался ранее, то мы уже посчитали для него количество
            повторений и вывели их на экран. Если же выбранное значение ранее не
            встречалось, то нужно посчитать для него количество повторений. */
            if (met_before)continue; //если элемент считался ранее, значит мы его уже
            считали. Далее код считать не надо
            int count = 0;
            for (int j = i + 1; j < n; j++)
            {
                if (arr[i] == arr[j]) count++;
            }
            if (count > 0)
                System.out.println("Значение %d повторяется %d раз".formatted(arr[i],

```

```
count)); //счетчик
    }
}
}
```

2) с клавиатуры вводится десятичное число и на экране появляется это же число в 16-ричной системе счисления (Hexadecimal) - каждая цифра отображается одним символом. буквы от А до F.

```
import java.util.Scanner;
import java.util.Arrays;
import java.util.Random;
import java.util.HashSet;

public class Main {
    public static void main(String[] args) {

        System.out.println("2");
        Scanner kb = new Scanner(System.in);
        System.out.println("Введите десятичное число: ");
        int decimal = kb.nextInt();
        int i = 0;
        char[] hexadecimal = {'0', '1', '2', '3', '4', '5', '6', '7',
                              '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
        String hexdecnum = "";

        while(decimal > 0) {
            i = decimal % 16;
            decimal /= 16;
            hexdecnum = hexadecimal[i] + hexdecnum;
        }
        System.out.print(hexdecnum);
    }
}
```

или так

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        for (int i=0; i<256; i++)
            System.out.println(i + "\t" +(char)i);

        Scanner kb = new Scanner(System.in);
        System.out.print(" Введите десятичное число: ");
        int decimal = kb.nextInt();
        char[] hex = new char [8];
        int i = 0;

        for (; decimal >0; decimal /=16) {
            hex[i] =(char)(decimal%16);
            if (hex[i] <10) hex[i] += 48;
            else          hex[i] += 55;
        }
    }
}
```

```

        i++;
    }
    for(--i; i>=0; i--)
        System.out.print(hex[i]);
    }
}

```

и так

```

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        for (int i=0; i<256; i++)
            System.out.println(i + "\t" +(char)i);

        Scanner kb = new Scanner(System.in);
        System.out.print(" Введите десятичное число: ");
        int decimal = kb.nextInt();
        char[] hex = new char [8];
        int i = 0;

        for (; decimal >0; decimal /=16; i++) {
            hex[i] =(char)(decimal%16);
            if (hex[i] <10) hex[i] += 48;
            else          hex[i] += 55;
        }
        for(--i; i>=0; i--)
            System.out.print(hex[i]);
        }
    }
}

```

Оператор “,” (comma operator) - позволяет в том месте где ожидается одно выражение написать несколько выражений. Эти выражения последовательно будут выполнены слева направо. И оператор запятая “,” вернет значения последнего выражения. Все выражения перечисленные через запятую должны возвращать значения одного типа.

```

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        for (int i=0; i<256; i++)
            System.out.println(i + "\t" +(char)i);

        Scanner kb = new Scanner(System.in);
        System.out.print(" Введите десятичное число: ");
        int decimal = kb.nextInt();
        char[] hex = new char [8];
        int i = 0;

        for (; decimal >0; decimal /=16, i++)
        {
            hex[i] = (char)(decimal%16);
            hex[i] = (char)(hex[i]<10 ? hex[i]+48 : hex[i]+55); // hex[i] += hex[i]<10 ?48:55;
        }
    }
}

```

```
}  
for(--i; i>=0; i--)  
    System.out.print(hex[i]);  
}  
}
```

Массивы

стр 55 java быстрый старт

Методы массивов

Методы, которые рассматриваются ниже, находятся в классе `java.util.Arrays`. Чтобы пользоваться ими, добавьте в свою программу команду `import java.util.Arrays`;

- ☐ **`equals()`** — метод проверяет равенство двух массивов. Если массивы равны, то метод возвращает `true`, а если нет — `false`. Два массива считаются равными, если они содержат одинаковое количество элементов, а эти элементы равны и следуют в одинаковом порядке.
- ☐ **`copyOfRange()`** — метод копирует содержимое одного массива в другой массив. При вызове он получает три аргумента.

Допустим, имеется следующий массив:

```
int [] source = {12, 1, 5, -2, 16, 14, 18, 20, 25};
```

Содержимое `source` можно скопировать в новый массив `dest` следующей командой:

```
int[] dest = Arrays.copyOfRange(source, 3, 7);
```

Первый аргумент (`source`) определяет массив с копируемыми значениями. Второй и третий аргументы сообщают компилятору, на каком индексе должно начинаться и останавливаться копирование соответственно. Иначе говоря, в нашем примере копируются элементы от индекса 3 до индекса 6 включительно (т. е. элемент с индексом 7 не копируется).

После копирования элементов метод `copyOfRange()` возвращает массив со скопированными числами. Этот массив присваивается `dest`.

Таким образом, массив `dest` после копирования содержит элементы `{-2, 16, 14, 18}`, тогда как массив `source` остается неизменным.

- ☐ **`toString()`** — метод возвращает объект `String`, представляющий элементы массива. Такое преобразование упрощает вывод содержимого массива.
Допустим, имеется массив

```
int[] numbers = {1, 2, 3, 4, 5};
```

Следующая команда может использоваться для вывода содержимого `numbers`.

```
System.out.println(Arrays.toString(numbers));
```

Команда выводит строку

```
[1, 2, 3, 4, 5]
```

в окне вывода

- ☐ **`sort()`** - метод предназначен для сортировки массивов. В аргументе ему передается массив.

Допустим, имеется массив

```
int [] numbers2 = {12, 1, 5, -2, 16, 14};
```

Чтобы отсортировать массив, выполните следующую команду:

```
Arrays.sort(numbers2);
```

Массив будет отсортирован по возрастанию.

Метод `sort()` не возвращает новый массив. Он просто изменяет массив, переданный

при вызове. Другими словами, он изменяет массив numbers2 в нашем примере. После этого можно воспользоваться командой `System.out.println(Arrays.toString(numbers2));` для вывода отсортированного массива. Команда выдает результат `[-2, 1, 5, 12, 14, 16]`

- ☐ **binarySearch()** - метод ищет конкретное значение в отсортированном массиве. Чтобы использовать этот метод, необходимо предварительно отсортировать массив. Для сортировки можно воспользоваться методом `sort()`, описанным выше.

Допустим, имеется следующий массив:
`int[] myInt = {21, 23, 34, 45, 56, 78, 99};`
Чтобы определить, присутствует ли значение 78 в массиве, выполните команду `int foundIndex = Arrays.binarySearch(myInt, 78);`
Значение `foundIndex` будет равно 5. Оно показывает, что число 78 находится в элементе массива с индексом 5.
С другой стороны, при выполнении команды `int foundIndex2 = Arrays.binarySearch(myInt, 39);` значение `foundIndex2` будет равно -4.
Этот результат состоит из двух частей — знака «-» и числа 4.
Знак «-» просто означает, что значение 39 не найдено.
С числом 4 дело обстоит сложнее. Оно показывает, где бы находилось данное число, если бы оно существовало в массиве. При этом индекс увеличивается на 1.
Другими словами, если бы число 39 присутствовало в массиве, то оно бы находилось в элементе с индексом $4 - 1 = 3$.
В этом разделе были рассмотрены некоторые часто используемые методы массивов. За полным списком всех методов массивов обращайтесь на страницу <https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html>.

Массивы

<https://javarush.com/quests/lectures/questsyntaxpro.level05.lecture00>

Массив — это скорее коробка, разделенная внутри на секции. У каждой секции в «коробке-массиве» есть ее номер. Нумерация, конечно же, с нуля..

Ну или можно провести еще одну аналогию. Давайте сравним обычный жилой дом и многоквартирный. Обычный дом занимает одна семья, а многоквартирный разбит на квартиры. Чтобы написать письмо семье, которая живет в обычном доме, надо указать его уникальный адрес. А чтобы написать письмо семье, которая живет в квартире, надо указать уникальный адрес дома и еще номер квартиры.

Так вот, переменная-массив — это переменная-многоэтажка. В ней можно хранить не одно значение, а несколько. В такой переменной есть несколько квартир (ячеек), к каждой из которых можно обратиться по ее номеру (индексу).

Для этого после имени переменной в квадратных скобках надо указать индекс ячейки, к которой обращаемся. Это довольно просто:

`array[индекс] = значение;`

Где **array** — это имя переменной-массива, **индекс** — номер ячейки в массиве, а **значение** — значение, которое мы хотим занести в указанную ячейку.

`int[] array = new int[100];` - хранить 100 целых чисел

`double[] vals = new double[20];` - массив на 20 ячеек для хранения вещественных чисел

В методе main проинициализируй переменные intArray и doubleArray массивами соответствующих типов, размер которых равен 10.

```
public class Solution {  
    public static void main(String[] args) {  
        int[] intArray = new int[10];  
        double[] doubleArray = new double[10];  
    }  
}
```

Код	Пояснение
<pre>int[] a = new int[10]; a[2] = 4; a[7] = 9; a[9] = a[2] + a[5];</pre>	<p>Создаем массив на 10 элементов типа int.</p> <p>В ячейку с индексом 2 записываем значение 4.</p> <p>В ячейку с индексом 7 записываем значение 9.</p> <p>В ячейку с индексом 9 записываем сумму значений, которые хранятся в ячейках 2 (хранится 4) и 5 (хранится 0).</p>

Важно. Все ячейки массива имеют одинаковый тип данных. Если мы создали массив строк String, в его ячейках можно хранить только строки. Тип данных массива задается при его создании. Ни тип данных, ни длину массива в дальнейшем поменять нельзя.

В методе main(String[]) тебе нужно заполнить массив strings значениями. Если индекс массива чётный — присвоить значение "Чётный" (ноль — цифра чётная), иначе присвоить "Нечётный". Вывод текста в консоль в тестировании не участвует.

```
public class Solution {  
    public static final String ODD = "Нечётный";  
    public static final String EVEN = "Чётный";  
    public static String[] strings = new String[5];  
  
    public static void main(String[] args) {  
        for(int i = 0; i < strings; i++)  
        {  
            strings[i] = (i%2==0) ? EVEN : ODD;  
        }  
        System.out.print("index = 0");  
        System.out.println(" value = " + strings[0]);  
        System.out.print("index = 1");  
        System.out.println(" value = " + strings[1]);  
        System.out.print("index = 2");  
        System.out.println(" value = " + strings[2]);  
        System.out.print("index = 3");  
        System.out.println(" value = " + strings[3]);  
        System.out.print("index = 4");  
        System.out.println(" value = " + strings[4]);  
    }  
}
```

Реализуй метод main(String[]), который меняет знак элемента массива array на противоположный, если значение этого элемента чётное

```

public class Solution {
    public static int[] array = new int[]{-1, 2, 3, -4, -5};
    public static void main(String[] args) {
        for(int i = 0; i < array; i++)
        {
            array[i] = array[i]%2==0 ? -array[i] : array[i];
        }
        System.out.println(array[0]);
        System.out.println(array[1]);
        System.out.println(array[2]);
        System.out.println(array[3]);
        System.out.println(array[4]);
    }
}

```

Код	Пояснение
<pre> int[] a = new int[10]; a[2] = 4; a[7] = 9; int[] b = a; a[9] = b[2] + a[7]; </pre>	<p>Создаем массив на 10 элементов типа int. В ячейку с индексом 2 записываем значение 4. В ячейку с индексом 7 записываем значение 9. В переменную b сохраняем адрес, который есть в переменной a. Теперь a и b указывают на один и тот же объект-массив в памяти. В ячейку с индексом 9 объекта-массива записываем сумму значений, которые хранятся в ячейках 2 (хранится 4) и 7 (хранится 9).</p>

Код	Пояснение
<pre> int n = 100; int[] a = new int[n]; a[n-1] = 2; a[n-2] = 3; a[n/5] = a[n-1] + a[n-2] </pre>	<pre> // a[99] = 2; // a[98] = 3; // a[20] = a[99] + a[98]; </pre>

Важно: Кстати, обращаем ваше внимание, что если попробовать обратиться к ячейке массива по индексу, которого в массиве нет (в нашем случае это все целые числа, кроме чисел 0..99), программа аварийно завершится с ошибкой `ArrayIndexOutOfBoundsException` — индекс за границами массива.

Длина массива

Можно отдельно создать переменную типа массив и потом где-то в коде присвоить ей значение (ссылку на объект-массив). Можно сделать даже так:

Код	Пояснение
-----	-----------

<pre>int[] array; if (a < 10) array = new int[10]; else array = new int[20];</pre>	<p>Создаем переменную-массив типа int[] Если переменная a меньше 10, то создать массив из 10 элементов. Иначе создать массив из 20 элементов</p>
---	---

Для этого у массива есть специальное свойство (переменная) — length. И узнать длину массива можно с помощью такого выражения:

array;

Вот как можно продолжить предыдущий пример

Код	Пояснение
<pre>int[] array; if (a < 10) array = new int[10]; else array = new int[20]; for (int i = 0; i < array; i++) { System.out.println(array[i]); }</pre>	<p>Создаем переменную-массив типа int[] Если переменная a меньше 10, то создать массив из 10 элементов. Иначе создать массив из 20 элементов Цикл по всем элементам массива: от 0 и до длины array — 1</p>

String[] list – это просто объявление переменной: сам контейнер(объект-массив) еще не создан. Чтобы с ним можно было работать, нужно создать массив (контейнер) и положить его в эту переменную, а потом уже им пользоваться.
Когда мы создаём объект-массив (контейнер), нужно указать, какой он длины — сколько в нем ячеек. Это делается командой вида: new TypeName[n];

Код	Пояснение
<pre>String s; String[] list;</pre>	<p>s равно null list равно null</p>
<pre>list = new String[10]; int n = list;</pre>	<p>Переменная list хранит ссылку на объект – массив строк из 10 элементов. n равно 10</p>
<pre>list = new String[0];</pre>	<p>Теперь list содержит массив из 0 элементов. Массив есть, но хранить элементы он не может.</p>
<pre>list = null; System.out.println(list[1]);</pre>	<p>Будет сгенерировано исключение (ошибка программы) — программа аварийно завершится. list содержит пустую ссылку — null</p>

```
list = new String[10];
System.out.println(list[10]);
```

Будет сгенерировано исключение (ошибка программы) — выход за границы массива. Если `list` содержит 10 элементов/ячеек, то их разрешённые индексы: 0 1 2 3 4 5 6 7 8 9 — всего 10 штук.

Реализовать метод `main(String[])`, в котором нужно скопировать содержимое массивов `firstArray` и `secondArray` в один массив `resultArray`. Массив `firstArray` должен быть в начале нового массива `resultArray`, а `secondArray` — после него.

```
/*
Объединяем массивы
*/
public class Solution {
    public static int[] firstArray = new int[]{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    public static int[] secondArray = new int[]{10, 11, 12, 13, 14, 15, 16, 17, 18, 19};
    public static int[] resultArray;

    public static void main(String[] args) {
        resultArray = new int[firstArray.length + secondArray.length];
        for (int i = 0; i < resultArray.length; i++) {
            if (i < firstArray.length) {
                resultArray[i] = firstArray[i];
            } else {
                resultArray[i] = secondArray[i - firstArray.length];
            }
            System.out.print(resultArray[i] + " ");
        }
    }
}
```

Примеры использования массивов

<https://javarush.com/quests/lectures/questsyntaxpro.level05.lecture02>

Заполнение массива из 10 чисел числами от 0 до 9:

```
int[] array = new int[10];
for (int i = 0; i < 10; i++) {
    array[i] = i;
}
```

- Создаем объект-массив на 10 элементов
- Цикл от 0 до 9 включительно
- В ячейки заносим значения от 0 до 9

Заполнение массива из 10 чисел числами от 1 до 10:

<pre>int[] array = new int[10]; for (int i = 0; i < 10; i++) { array[i] = i + 1; }</pre>	<ul style="list-style-type: none"> • Создаем объект-массив на 10 элементов • Цикл от 0 до 9 включительно • В ячейки заносим значения от 1 до 10
---	--

Заполнение массива из 10 чисел числами от 10 до 1:

<pre>int[] array = new int[10]; for (int i = 0; i < 10; i++) { array[i] = 10 - i; }</pre>	<ul style="list-style-type: none"> • Создаем объект-массив на 10 элементов • Цикл от 0 до 9 включительно • В ячейки заносим значения от 10 до 1
--	--

Вывод чисел в обратном порядке

<pre>Scanner console = new Scanner(System.in); int[] array = new int[10]; for (int i = 0; i < 10; i++) { array[i] = console.nextInt(); } for (int i = 9; i >= 0; i--) { System.out.println(array[i]); }</pre>	<ul style="list-style-type: none"> • Создаем объект Scanner • Создаем объект-массив на 10 элементов • Цикл от 0 до 9 включительно • Читаем число с клавиатуры и сохраняем его в очередную ячейку массива • Цикл от 9 до 0 включительно • Выводим на экран очередную ячейку массива
---	--

Нужно написать программу, которая:

1. Считывает с консоли целое число N.
2. Если считанное число N больше 0, то программа дальше считывает целые числа, количество которых равно N.
3. Вывести в консоль считанные числа, если N нечетное — в порядке ввода, иначе — в обратном порядке.

Каждое число выводить с новой строки. Число N выводить не нужно.

```
import java.util.Scanner;
```

```
/*
```

```
Reverse
```

```
*/
```

```
public class Solution {
```

```
    public static void main(String[] args) {
```

```
        Scanner console = new Scanner(System.in);
```

```
        int input = console.nextInt(); //Считывает с консоли целое число N.
```

```
        if (input > 0) { //Если считанное число N больше 0
```

```
            int[] array = new int[input];
```

```
            for (int i = 0; i < input; i++) {
```

```
array[i] = console.nextInt(); //Вывести в консоль считанные числа  
}  
if (input % 2 == 0) { //  
    for (int i = 0; i < array.length; i++) { //иначе — в обратном порядке.  
        System.out.println(array[array.length - i - 1]);  
    }  
} else {  
    for (int i = 0; i < array.length; i++) { //если N нечетное — в порядке ввода  
        System.out.println(array[i]);  
    }  
}  
}  
}  
}
```

Чтобы найти минимальный элемент нужно:

- Взять первый элемент массива в качестве «текущего найденного минимального».
- Сравнить с ним все элементы массива поочередно
- Если очередной элемент массива меньше «текущего минимального», то обновить значение «текущего минимального элемента»

Поиск минимального элемента в массиве

```
Scanner console = new
Scanner(System.in);
int[] array = new int[10];
for (int i = 0; i < 10; i++)
{ array[i] =
console.nextInt();
}
int min = array[0];

for (int i = 1; i < 10; i++) {
    if (array[i] < min)
        min = array[i];
}
System.out.println(min);
```

- Создаем объект Scanner
- Создаем объект-массив на 10 элементов
- цикл от 0 до 9 включительно
- Читаем число с клавиатуры и сохраняем его в очередную ячейку массива
- В качестве минимального числа взяли нулевой элемент массива
- Цикл от 1 до 9 включительно
- Если текущий элемент массива меньше «найденного минимального числа»
- то «обновить значение минимального числа»
- Вывести найденное минимальное число на экран

```

Минимальное min из N чисел
Чтобы выполнить эту задачу, тебе нужно:
1. Ввести с клавиатуры число N.
2. Считать N целых чисел и заполнить ими массив.
3. Найти минимальное число среди элементов массива и вывести в консоль
import java.util.Scanner;
/*
Минимальное из N чисел
*/
public class Solution {
    public static int[] array;

```

Чтобы выполнить эту задачу, тебе нужно:

1. Ввести с клавиатуры число N.
2. Считать N целых чисел и заполнить ими массив.
3. Найти минимальное число среди элементов массива и вывести в консоль

```
import java.util.Scanner;
```

 I^*

Минимальное из N чисел

*/

```
public class Solution {
    public static int[] array;
```

```

public static void main(String[] args) throws Exception {
    Scanner scanner = new Scanner(System.in);
    int n = Integer.parseInt(scanner.nextLine()); //Ввести с клавиатуры число N.

    array = new int[n]; //Считать N целых чисел
    for (int i = 0; i < n; i++) {
        array[i] = Integer.parseInt(scanner.nextLine()); // заполнить ими массив.
    }

    int min = array[0]; //Найти минимальное число
    for (int i = 1; i < array.length; i++) {
        int number = array[i];
        if (number < min) {
            min = number;
        }
    }
    System.out.println(min); //вывод в консоль
}
}

```

Максимальное max из N чисел

В этой задаче тебе нужно: 1. Ввести с клавиатуры число N. 2. Считать N целых чисел и заполнить ими массив array. 3. Найти максимальное число среди элементов массива.

```
import java.util.Scanner;
```

```
/*
```

```
Максимальное из N чисел
```

```
*/
```

```
public class Solution {
```

```
    public static int[] array;
```

```
    public static void main(String[] args) throws Exception {
```

```
        Scanner scanner = new Scanner(System.in);
```

```
        int n = Integer.parseInt(scanner.nextLine()); //Ввести с клавиатуры число N.
```

```
        array = new int[n]; //Считать N целых чисел
```

```
        for (int i = 0; i < n; i++) {
```

```
            array[i] = Integer.parseInt(scanner.nextLine()); // заполнить ими массив array.
```

```
        }
```

```
        int max = array[0]; //Найти максимальное число
```

```
        for (int i=0; i<n; i++)
```

```
        {
```

```
            if(array[i] > max) max = array[i];
```

```
        }
```

```
        System.out.println(max); //вывод в консоль
```

```
    }
```

```
}
```

Продвинутая работа с массивами

<https://javarush.com/quests/lectures/questsyntaxpro.level05.lecture03>

Массив типа String

Вводит с клавиатуры 10 строк и выводит их на экран в обратном порядке.

```
Scanner console = new
Scanner(System.in);
String[] array = new
String[10];
for (int i = 0; i < 10; i++) {
array[i] =
console.nextLine(); }
for (int i = 9; i >= 0; i--) {
System.out.println(array[i]);
}
```

- Создаем объект Scanner
- Создаем объект-массив на 10 элементов
- Цикл от 0 до 9
- Читаем строку с клавиатуры и сохраняем ее в очередную ячейку массива
- Цикл от 9 до 0
- Выводим на экран очередную ячейку массива

Нужно: 1) Считать 6 строк и заполнить ими массив strings.

2) Удалить повторяющиеся строки из массива strings, заменив их на null (null должны быть не строками "null").

Scanner console = new Scanner(System.in); // Создаем объект класса Scanner для ввода строк с клавиатуры

strings = new String[6]; //присваиваем массиву strings длину 6 строк

for(int i = 0; i < strings.length; i++) { //Запускаем цикл по всем строкам массива от нуля до конца

strings[i] = console.nextLine(); // Каждому элементу массива присваивается

значение введенное через клавиатуру

}

for(int i = 0; i < strings.length; i++) { //Запускаем цикл по всем строкам массива, чтобы взять "первый" он же нулевой элемент массива.

String iString = strings[i]; // первый элемент массива кладем в коробку/переменную iString;

for (int j=i+1;j< strings.length;j++) { // Запускаем второй цикл внутри первого //Этот цикл берет второй элемент,(строку) массива (j=i+1 или j=0+1=1=второй элемент) начальным, и идет по всем строкам массива параллельно с первым циклом.

//Опережая первый цикл на одну строку.

if (iString == null) { // Создаем условие, если в коробке переменной iString ничего нет break;} //цикл прерываем

else if (iString.equalsIgnoreCase(strings[j])) { //если что-то есть, и сравнение по строкам переменной iString равно содержимому некст ячейки массива strings[i]=null; //обнуляем первую ячейку которую мы захватили

первым циклом

strings[j]=null; //обнуляем вторую ячейку которую мы захватили

вторым циклом

}}}

for(int i = 0; i < strings.length; i++) { // Снова Запускаем цикл по всем строкам массива от нуля до конца

System.out.print(strings[i] + " , "); //Выводим содержимое массива на экран

}

}

}

GK__Java_urok_05

Массив в Java – это конечный набор элементов одного типа. Массив в Java это объект, поэтому переменная, указывающая на массив, является переменной ссылочного типа.

Синтаксис объявления ссылки массива:

```
тип[ ] идентификатор;  
или  
тип идентификатор[ ].
```

Объект массива создается с помощью оператора new.

Синтаксис создания массива элементов:

```
new int[<количество элементов>];
```

ВАЖНО!!! Сразу после создания массива его элементы инициализируются значениями по умолчанию, для всех числовых типов это 0: для char – символ с юникодом 0, для boolean – false, для ссылочных типов – null.

Обращение к элементам массива происходит по индексу. Индексация элементов массива начинается с 0.

Значение элементов массива можно установить в момент создания массива. Этот процесс называется **инициализация массива**.

```
int[] mas = new int[]{3, 2, 1}; // будет создан массив  
// на три элемента, его первый элемент будет иметь значение 3, второй 2, третий 1.
```

Внимание!!! Нельзя одновременно инициализировать массив и указывать его размер при создании, так как его размер задается количеством элементов в блоке инициализации.

Массив имеет свойство length, в котором хранится длина массива (количество элементов заданных при создании массива), изменить это свойство нельзя.

```
int[] mas = {3, 2, 1};  
mas.length = 5; // ошибка компиляции  
System.out.println(mas.length);  
mas = {1, 2, 3, 4, 5}; // ошибка компиляции, можно использовать только при  
объявлении.
```

<https://www.sololearn.com/>

Массив хранит несколько значений в одной переменной. Чтобы объявить массив, вам нужно определить тип элементов в квадратных скобках.

```
int[ ] ages;
```

Имя массива - ages. Предназначен тип элементов, которые он будет содержать. Теперь, чтобы создать массив, нам нужно указать количество элементов, которые он будет содержать, используя ключевое слово new: Доступ к элементам массива осуществляется с помощью их индексов, заключенных в квадратные скобки. Первый элемент имеет индекс 0.

```
int[ ] ages; - указать тип элементов
```

ages = new int[5];- указываем кол-во элементов

или

int[] ages = new int[5];

Пример

```
String[] names = { "A", "B", "C", "D"};
```

```
System.out.println(names[2]);
```

вывод C

Можно написать так String[] menu = new String[] {"Tea", "Espresso", "Americano"}; или так String[] menu = {"Tea", "Espresso", "Americano"};

```
import java.util.SimpleTimeZone;
public class Main {
    public static void main(String[] args) {
        String[] menu = {"Tea", "Espresso", "Americano", "Water", "Hot Chocolate"};
        Scanner sc = new Scanner(System.in);
        System.out.println("Число: ");
        int i = sc.nextInt();
        if (i <= -1 || i >= 5) {
            System.out.println("Invalid");
        } else {
            System.out.println(menu[i]);
        }
    }
}
```

Чтобы использовать цикл, нам сначала нужно выяснить, сколько элементов хранится в массиве. Для этого массив имеет свойство length, доступ к которому осуществляется следующим образом:

```
int[] ages = {18, 33, 24, 64, 45};
```

```
System.out.println(ages.length); //Это выведет количество элементов, хранящихся в массиве.
```

Вывести все элементы массива

```
public class Demo {
    public static void main(String[] args) {
        int[] ages = {18, 33, 24, 64, 45};

        for(int x=0;x<ages.length;x++) {
            System.out.println(ages[x]);
        }
    }
}
```

ВЫЧИСЛИМ СУММУ ВСЕХ ЗНАЧЕНИЙ В МАССИВЕ

```
int[] ages = {18, 33, 24, 64, 45};
int sum = 0;
for(int x=0;x<ages.length;x++) {
    sum += ages[x];
}
System.out.println(sum);
```

Вычислить сумму элементов двойного массива, называемого nums, используя цикл for, затем выведите ее на экран.

```
double sum = 0.0;
for (int x=0; x<nums.length; x++) {
```

```
sum += nums[x];
}
System.out.println(sum);
```

Данный код объявляет массив, в котором хранятся ежемесячные доходы компании за год. Вам нужно рассчитать среднемесячный доход за год. Для этого вычислите сумму выручки за все месяцы и разделите ее на количество элементов в массиве. Вы можете найти количество элементов в массиве, используя свойство `length`. Поскольку массив имеет тип `double`, выведите результат в виде `double`.

```
public class Program {
    public static void main(String[] args) {
        double[] revenue = {88750, 125430, 99700, 14500, 158000, 65000, 99000, 189000,
        210000, 42000, 165800, 258900};
        //your code goes here
        double sum = 0;
        for (int x = 0; x < revenue.length; x++) {
            sum += revenue[x];
        }
        System.out.println(sum/revenue.length);
    }
}
```

Двумерные массивы (07.09.2023)

Двумерный массив - это таблица значений как и у любой другой таблицы у двумерного массива есть строки столбики. Как строки так и элементы строки двумерного массива нумеруются с нуля.

Для двумерного массива как правило используется вложенный `for`.

```
import java.util.Random;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.println("Введите количество строк: ");
        int rows = kb.nextInt();
        System.out.println("Введите количество элементов столбиков: ");
        int cols = kb.nextInt();

        int[][] arr = new int[rows][cols]; //всегда вначале строки, потом столбики

        Random rand = new Random(0);
        for (int i=0; i<rows; i++)//вложенный for
        {
            for (int j=0; j<cols; j++)
            {
                arr[i][j]=rand.nextInt(100);
            }
        }
        for(int i=0; i < rows; i++)
        {
            for (int j=0; j<cols; j++)
            {
```

```
        System.out.print(arr[i][j] + "\t");
    }
    System.out.println();
} } }
```

или так

```
import java.util.Random;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Введите количество строк: ");
        int rows = kb.nextInt();
        System.out.print("Введите количество элементов столбиков: ");
        int cols = kb.nextInt();

        int[][] arr = new int[rows][cols]; // всегда вначале строки, потом столбики

        Random rand = new Random(0);
        for (int i = 0; i < rows; i++) // вложенный for
        {
            for (int j = 0; j < cols; j++)
            {
                arr[i][j] = rand.nextInt(100);
            }
        }
        System.out.println("Количество строк: " + arr.length);
        System.out.println("Количество элементов строки: " + arr[0].length); // взяли
        // нулевую строку и взяли ее длину
        for (int i = 0; i < arr.length; i++)
        {
            for (int j = 0; j < arr[i].length; j++)
            {
                System.out.print(arr[i][j] + "\t");
            }
            System.out.println();
        }
    }
}
```

Д/з18 <https://github.com/OlesyaVolkova/DZ18>

1) Для двумерного массива найти: числа псевдослучайные

- сумму элементов
- среднее-арифметическое элементов
- минимальное значение
- максимальное значение
- отсортировать двумерные массив через алгоритм (не SORT) СЛОЖНАЯ

```
class Main {
    public static void main(String[] args) {

        int[][] arr = {
            { 1, 2, 3 },
            { 4, 5, 6 },
        }
```

```

        { 7,8,9 }
    };

    int sum = 0;
    for (int i = 0; i<arr.length;i++) {
        for (int j = 0; j<arr[i].length;j++) {
            sum += arr[i][j];
        }
    }
    System.out.println("Сумма элементов: " + sum);

    int count = 0;
    for (int x = 0; x<arr.length;x++) {
        for (int y = 0; y<arr[x].length;y++) {
            sum += arr[x][y];
            count++;
        }
    }
    System.out.println("Среднее-арифметическое элементов: " + (double)sum/count);

    int min, max;
    max = Integer.MIN_VALUE;
    min = Integer.MAX_VALUE;
    for (int a = 0; a<arr.length; a++) {
        for (int b = 0; b < arr[a].length; b++) {
            if (arr[a][b] < min) min = arr[a][b];
            if (arr[a][b] > max) max = arr[a][b];
        }
    }
    System.out.println("Минимальное значение: " + min);
    System.out.println("Максимальное значение: " + max);

    // сортировать двумерные массивы через алгоритм
    for (int i=0; i<arr.length; i++){
        for (int j=0; j<arr[i].length; j++){
            for (int k=0; k<arr.length; k++)
            {
                for (int l=0; l< arr[k].length; l++){
                    //arr[i][j] выбранный элемент
                    //arr[k][l] перебираемый элемент
                    if (arr[k][l] < arr[i][j]) //если < сортировка по убыв, если > то возрастанию
                    {
                        int buffer = arr[i][j];
                        arr[i][j] = arr[k][l];
                        arr[k][l] = buffer;
                    }
                }
            }
        }
    }
    for (int i=0; i<arr.length; i++) System.out.println(Arrays.toString(arr[i]));
    /*второй вариант
    int iter = 0;
    int ex = 0;

```

```

        for (int i=0; i<arr.length; i++){
            for (int j=0; j<arr[i].length; j++){
                //arr[i][j]выбранный элемент
                for (int k=i; k<arr.length; k++) {
                    for (int l=k == i? j:0; l< arr[k].length; l++){
                        //arr[i][j]выбранный элемент
                        //arr[k][l]перебираемый элемент
                        iter++;
                        if (arr[k][l] < arr[i][j]) //если < сортировка по убыванию, если > то
возрастание
                        {
                            int buffer = arr[i][j];
                            arr[i][j] = arr[k][l];
                            arr[k][l] = buffer;
                            ex++;
                        }
                    }
                }
            }
        }
        System.out.println("Массив отсортирован за %d итераций, при этом сделано %d
замен элементов".formatted(iter,ex));
        for (int i=0; i<arr.length; i++) System.out.println(Arrays.toString(arr[i]));
    */
    }
}

```

2) Заполнить двумерный массив уникальными случайными числами

```

import java.util.Random;

class Main {
    public static void main(String[] args) {

        int rows = 3;
        int cols = 4;
        int[][] arr= new int[rows][cols];

        Random rand = new Random(0);
        for (int i=0; i<rows; i++)
        {
            for (int j=0; j<cols; j++)
            {
                int newRand;
                newRand = rand.nextInt(12);
                arr[i][j] = newRand;
            }
        }
        for(int i=0; i < rows; i++) { // цикл по строкам для вывода массива
            for (int j = 0; j < cols; j++) { // цикл по столбцам для вывода массива
                System.out.print(arr[i][j] + " "); // вывод элемента массива
            }
            System.out.println(); // переход на новую строку
        }
    }
}

```

или так


```
import java.util.Arrays;
import java.util.Random;

class Main {
    public static void main(String[] args) {
        Random rand = new Random(0);
        int rows = 3;
        int cols = 4;
        int[][] arr = new int[rows][cols];
        System.out.println("Start");
        for(int i=0; i<arr.length; i++)
        {
            for(int j=0; j<arr[i].length; j++)
            {
                boolean unique;
                do
                {
                    unique = true;
                    arr[i][j]=rand.nextInt(rows*cols);
                    for(int k=0; k<=i; k++)
                    {
                        for(int l=0; l < (k == i ? j:arr[k].length); l++){
                            if(arr[i][j]==arr[k][l])
                            {
                                unique = false;
                                break;
                            }
                        }
                    }
                    if(!unique)break;
                } while(!unique);
            }
        }
        for (int i=0; i<arr.length; i++)
            System.out.println(Arrays.toString(arr[i]));
    }
}
```

3) В двумерном массиве заполненном случайными числами найти количество повторений каждого значения (решали на первой паре, только делали на одномерном массиве).

```
int rows = 3;
int cols = 3;
int[][] arr= new int[rows][cols];

Random rand = new Random();
for (int i=0; i<rows; i++)
{
    for (int j=0; j<cols; j++)
    {
        int newRand;
        newRand = rand.nextInt(100);
        arr[i][j] = newRand;
    }
}
for(int i=0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        System.out.print(arr[i][j] + "\t");
    }
    System.out.println();
}

for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        int count = 0;
        for (int k = 0; k < rows; k++) {
            for (int l = 0; l < cols; l++) {
                if (arr[i][j] == arr[k][l]) {
                    count++;
                }
            }
        }
        if (count >1) System.out.println("Число " + arr[i][j] + " встречается " + count + "
раз");
    }
}
}
```

Двумерные массивы

<https://javarush.com/quests/lectures/questsyntaxpro.level05.lecture04>

Ячейки массива можно представить не только в виде столбца (или строки), но и в виде прямоугольной таблицы.

```
int[][] имя = new int[ширина][высота];
```

Где **ИМЯ** — это имя переменной-массива, ширина — это **ширина** таблицы (в ячейках), а **ВЫСОТА** — это высота таблицы

```
int[][] data = new int[2][5];
data[1][1] = 5;
```

Создаем двумерный массив: два столбца и 5 строк.
В ячейку (1,1) записываем 5.


```
// длины месяцев года поквартально
int[][] months = { {31, 28, 31}, {30, 31, 30}, {31, 31, 30}, {31, 30, 31} };
```

Есть очень много мест, где вам как программисту может понадобиться двумерный массив. Реализация практически любой настольной игры — это же готовый двумерный массив: «Шахматы», «Шашки», «Крестики-Нолики», «Морской бой»:

Когда мы создаем массив `new int[2][5]`; у нас таблица «две строки и 5 столбцов» или все-таки «два столбца и 5 строк»?

```
// Важная матрица с данными
int[][] matrix = { {1, 2, 3, 4, 5}, {1, 2, 3, 4, 5} };
```

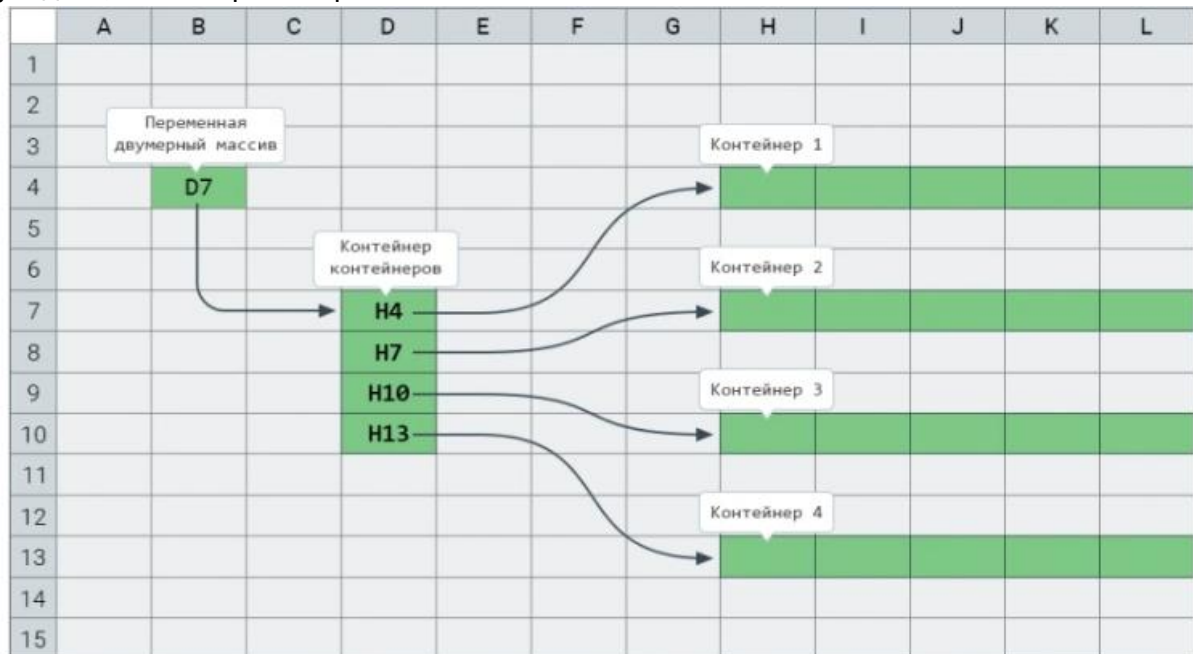
Ничего не замечаете? А если так:

```
// Важная матрица с данными
```

```
int[][] matrix = {
    {1, 2, 3, 4, 5},
    {1, 2, 3, 4, 5}
};
```

Если мы напишем наши данные в коде построчно, то получим таблицу, у которой 2 строки и 5 столбцов.

Двумерные массивы — это на самом деле массивы массивов! Другими словами, если в случае с обычным массивом «переменная-массив хранит ссылку на контейнер, который хранит элементы массива». То в случае с двумерными массивами у нас ситуация немного взрывоопаснее: переменная-двумерный-массив хранит ссылку на контейнер, который хранит ссылки на одномерные массивы. Это лучше один раз увидеть, чем сто раз попробовать объяснить:



Слева у нас «переменная-двумерный-массив», которая хранит ссылку на «объект-двумерный массив». В середине у нас «объект двумерный массив», в ячейках которого хранятся ссылки на одномерные массивы — строки двумерного массива. Ну и справа вы видите четыре одномерных массива — строки нашего двумерного массива.

можно поменять строки местами

```
int[][] matrix = {
    {1, 2, 3, 4, 5},
```

Двумерный массив
В `matrix[0]` у нас хранится ссылка на первую

<pre> {5, 4, 3, 2, 1} }; int[] tmp = matrix[0]; matrix[0] = matrix[1]; matrix[1] = tmp; </pre>	<p>строку. Меняем ссылки местами. В итоге массив matrix выглядит так:</p> <pre> { {5, 4, 3, 2, 1}, {1, 2, 3, 4, 5} }; </pre>
---	--

Проинициализируй массив MULTIPLICATION_TABLE значением new int[10][10], заполни его таблицей умножения и выведи в консоли в следующем виде:

```

1 2 3 4 ...
2 4 6 8 ...
3 6 9 12 ...
4 8 12 16 ...
...

```

Числа в строке разделены пробелом.

/*

Таблица умножения

*/

```

public class Solution {
    public static int[][] MULTIPLICATION_TABLE;
    public static void main(String[] args) {
        MULTIPLICATION_TABLE = new int[10][10]; //инициализируем двумерный
        массив 10 x 10
        for (int w = 0; w < 10; w++) { // первый цикл шагает по ширине массива
            for (int h = 0; h < 10; h++) { // второй цикл шагает по высоте массива
                MULTIPLICATION_TABLE[w][h] = (w + 1) * (h + 1); // формула заполнения
                массива правильным числом (попробуйте в ручную подставить цифры так поймете
                быстрее)
                System.out.print(MULTIPLICATION_TABLE[w][h]+" "); // выводим полученное
                значение через пробел
            }
            System.out.println(); // делаем вывод с новой строки еще внутри цикла на
            ширину, чтобы разделить строки
        }
    }
}

```

/* MULTIPLICATION_TABLE = new int[10][10]; - это массив, в котором лежат 10 массивов. В каждом из 10 массивов лежат 10 циферок.

Автоматическое заполнение происходит через вложенные циклы. Т.е. выполняется цикл, внутри которого выполняется ещё один цикл.

Первый (основной) цикл выбирает с каким из 10-ти массивов мы будем работать.

Второй (вложенный) цикл выбирает индекс числа, с которым мы будем работать уже внутри выбранного массива.

Получается, основной цикл выбирает массив с индексом [0], вложенный цикл проходит по этому массиву и заполняет его нужными нам циферками, затем основной цикл переходит к следующему массиву с индексом [1] и всё повторяется заново, до тех пор, пока не заполнится весь двумерный массив.*/

Многомерные массивы

<https://www.sololearn.com/>

Массив создается с использованием двух квадратных скобок, указывающих на двумерность. Чтобы получить доступ к элементу в двумерном массиве, укажите два индекса: один для массива, а другой для элемента внутри этого массива.

```
public class Demo {
    public static void main(String[] args) {
        int[][] sample = { {1, 2, 3}, {4, 5, 6} }; //обращается к первому элементу во втором массиве sample:
        int x = sample[1][0];
        System.out.println(x);
    }
}
```

Выведите 2-й элемент 3-й строки.

```
int [][] grid = {{8,4}, {2,5}, {9,6}}; //(6) Строки | столбцы C0, C1 R0: 8 4 R1: 2 5 R2: 9 6
int x = grid [2][1];
System.out.println(x); //ответ 6
```

Два индекса массива называются индексом строки и индексом столбца. Вот как мы можем это визуализировать:

```
public class Demo {
    public static void main(String[] args) {
        int[][] sample = {
            {1, 2, 3},
            {4, 5, 6}
        };
        int x = sample[1][0];
        System.out.println(x);
    }
}
```

Чтобы выполнить цикл по двумерному массиву, нам нужны вложенные циклы for: первый цикл повторяется по строкам, а второй - по их элементам.

```
public class Demo {
    public static void main(String[] args) {
        int[][] sample = {
            {1, 2, 3},
            {4, 5, 6}
        };
        for(int x=0; x<sample.length; x++) {
            for(int y=0; y<sample[x].length; y++) {
                System.out.println(sample[x][y]);
            }
        }
    }
}
```

```

class Main {
    public static void main(String[] args) {
        int sum = 0;
        for(int x=0; x<arr.length; x++) {
            for(int y=0; y< arr[x].length; y++) {
                sum += arr[x][y];
            }
        }
        System.out.println(sum);
    }
}

```

Вы создаете билетную программу для небольшого кинотеатра. Сиденья представлены с использованием двумерного массива. Каждый элемент может иметь значения 1 и 0 - 1 занят, и 0, если он свободен. Ваша программа должна принимать в качестве входных данных строку и столбец места и выводить Free, если оно бесплатное, и Sold, если нет.

```

import java.util.Scanner;

public class Program {

    public static void main(String[] args) {
        int[][] seats = {
            {0, 0, 0, 1, 1, 1, 0, 0, 1, 1},
            {1, 1, 0, 1, 0, 1, 1, 0, 0, 0},
            {1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
            {0, 0, 0, 1, 1, 1, 1, 0, 0, 0},
            {0, 1, 1, 1, 0, 0, 0, 1, 1, 1}
        };
        Scanner sc = new Scanner(System.in);
        //your code goes here

        System.out.print("");
        int x = sc.nextInt();
        System.out.print("");
        int y = sc.nextInt();
        int v = seats[x][y];

        if (v == 1) {
            System.out.println("Sold");
        }
        else {
            System.out.println("Free ");
        }
    }
}

```

В массиве dates хранятся годы рождения группы людей. Вам нужно подсчитать, сколько из них родилось в 2000 году. Заполните пробелы, чтобы создать цикл, и вычислите результат.

```

class Main {
    public static void main(String[] args) {
        int count = 0;
        int[] dates;
        for(int x = 0; x<dates.length; x++) {
            if(dates[x] == 2000) {
                count ++;
            }
        }
        System.out.println(count);
    }
}

```

Ответ 2

```

int arr[ ] = new int[3];
for (int i = 0; i < 3; i++) {
    arr[i] = i;
}
int res = arr[0] + arr[2];
System.out.println(res);

```

```

public class Main {
    public static void main(String[] args) {
        int even = {2, 4, 6, 8};
        int res = even[0]+even[3];
        System.out.println(res);    } }

```

вычислить сумму всех элементов в массиве arr, используя цикл for-each:

```

int res = 0;
for (int el : arr) {
    res += el;
}

```

Многомерные массивы **GK_Java_urok_05**

Многомерных массивов в языке Java нет. Для имитации работы с данными в n-мерном пространстве в Java используется вложение массивов. Для создания многомерных массивов при объявлении добавляются дополнительные квадратные скобки.

Пример создания двумерного массива:

`int[][] mas = new int[5][5];` // создается массив на 5 элементов, где элементами являются другие массивы на 5 элементов.



Рис. Структура двумерного массива в Java

Пример создания трехмерного массива:

```
int[][][] mas = new int[5][3][3];
```

Обратиться к элементу многомерного массива можно, используя несколько индексов.

Пример записи значения в квадратный* массив:

```
int[][] mas = new int[3][3]; // создается массив на 5
```

```
mas[2][1] = 3; // запись значения в массив
```

Синтаксис многомерных массивов предусматривает отложенную инициализацию.

```
int[][] mas = new int[3][]; // создается одномерный массив из трех элементов, значения которых null
mas[0] = new int[4];
mas[1] = new int[]{3, 3};
mas[2][2] = 7; // ошибка во время выполнения
```

Использование цикла for-each для работы с массивами.

Пример вывода значений массива:

```
int [] numbers = new int[] { 3, 2, 1 };
for (int number : numbers)
{
    System.out.print(number);
}
```

Пример:

```
int [] numbers = new int[] { 3, 2, 1 };
for (int number : numbers)
{
    number = 3; // переменная number может быть использована только для чтения значений элементов массива
}
System.out.print(Arrays.toString(numbers))
```

Результат: [3, 2, 1]

Метод Java.lang.System.arraycopy()

Метод arraycopy копирует значения из одного массива в другой. Метод нативный* (написан на языке C++), и благодаря этому копирует значения из одного массива в другой быстрее любого кода написанного с использованием языка Java. Используется в коллекциях, например в Java.util.ArrayList.

Синтаксис:

```
System.arraycopy(src, srcPos, dest, destPos, length)
```

где **src** – массив, откуда копируются значения, **dest** – массив, куда копируются значения, **length** – количество копируемых элементов из массива **src**, **srcPos** – индекс в массиве **src**, с которого начинается копирование, **destPos** – индекс в массиве **dest**, в который начнут копироваться элементы.

```
int[] src = new int[]{1, 2, 3, 4, 5};
int[]      = new int[]{5, 4, 3, 2, 1};
int length = 3;
int srcIndex = 1;
int destIndex = 2;
```

```
System.arraycopy(src, srcIndex, destIndex, length);  
System.out.println(Arrays.toString( ));
```

Результат: [5, 4, 2, 3, 4]

Примечание. Если указанное количество элементов отсутствует в исходном массиве или в массиве –приемнике, случится исключительная ситуация.

Класс Java.util.Array

Класс Arrays содержит статические методы для работы с массивами.

■ **toString** – метод преобразует содержимое массива в строку. Если в качестве элементов массив содержит объекты, то для каждого элемента будет вызван метод toString() и полученные строки будут конкатенированы*.

Пример вывода содержимого массива в консоль:

```
int [] mas = new int[]{1,2,3};  
String content = Arrays.toString(mas);  
System.out.println(content );
```

Для вывода на консоль многомерных массивов необходимо использовать метод Arrays.deepToString().

Пример вывода на консоль содержимого многомерного массива:

```
String [][] objects = new String[3][3];  
String content = Arrays.deepToString(objects);  
System.out.println(content);
```

■ **fill** - метод заполняет массив одинаковыми значениями.

Пример заполнения элементов массива в диапазоне с 2 по 5 значением 2:

```
int[] mas = new int[10];  
int startIndex = 1;  
int endIndex = 4;  
Arrays.fill(mas, startIndex, endIndex, 2);
```

■ **sort** – сортирует значения массива в естественном порядке (от меньшего к большему).

```
int [] mas = new int[] {3, 1, 4, 6, 2};  
Arrays.sort(mas);  
System.out.println(Arrays.toString(mas));
```

Результат: [1, 2, 3, 4, 6]

Если в массиве хранятся элементы ссылочных типов, в методе sort есть возможность задать правила сортировки с использованием объекта класса реализующего интерфейс Comparator.

Пример сортировки по убыванию:

```
Integer [] mas = new Integer[] {3, 1, 4, 6, 2};  
Arrays.sort(mas, Collections.reverseOrder());
```

```
System.out.println(Arrays.toString(mas));  
Результат: [6, 4, 3, 2, 1]
```

■ **equals** – сравнивает содержимое двух массивов (поэлементно) и если количество элементов равно и все элементы эквиваленты, возвращает истину; иначе ложь.

```
int [] mas1 = {1,2,3};  
int [] mas2 = {1,2,3};  
boolean isEqual = Arrays.equals(mas1, mas2);  
System.out.println(isEqual);
```

■ **binarySearch** – возвращает индекс, под которым был найден элемент, используя бинарный поиск.

```
int index =Arrays.binarySearch(mas, 3);
```

■ **deepHashCode** – возвращает хэш-код, вычисленный на основе глубокого анализа элементов массива. Можно использовать для массивов, хранящих ссылочные типы.

■ **asList** – возвращает неизменяемый список (объект-адаптер), преобразуя элементы массива в элементы списка.

Зубчатые (неровные) массивы в Java

<https://javarush.com/quests/lectures/questsyntaxpro.level05.lecture05>

Для начала нужно создать «контейнер контейнеров» – первый массив, который будет хранить ссылки на массивы-строки. Делается это так:

```
int[][] имя = new int[высота][];
```

Вот как будет выглядеть итоговый код:

```
// Важная матрица с данными  
int[][] matrix = new int[2][];  
matrix[0] = new int[10];  
matrix[1] = new int[50];
```

Двумерный массив

Нулевая строка — массив из 10 элементов
Первая строка — массив из 50 элементов.

Кстати, как узнать длину «контейнера контейнеров» в нашем примере? Это ведь тоже объект-массив, а значит, у него есть длина. Правильный ответ — `matrix.length`. А у массивов-строк как? `matrix[0].length`

Строки GK__Java_urok_05

Класс `String` – это самый часто используемый класс в Java, он предназначен для хранения набора (массива) символов. Состояние объектов класса `String` невозможно изменить после создания объекта (объекты класса являются неизменяемыми). Следуя из объявления класса – `public final class String`, запрещено наследовать подклассы от класса `String`.

Класс String реализует три интерфейса: Serializable (сериализация), Comparable<String> (сравнение), CharSequence (последовательность символов). Класс String в сущности является оберткой над массивом символов (char [] value).

Внутреннее устройство класса String:

```
public final class String {  
    private final char value[]; // массив символов  
    private final int offset; // смещение от начала массива  
    private final int count; // количество символов в строке  
    private int hash; // hash код строки  
}
```

Создать объект строка в Java можно с помощью оператора new или строкового литерала (символы, ограниченные двойными кавычками).

Пример со строковым литералом:

```
String name = "Вася";  
name = ""; // пустая строка
```

Пример с оператором new :

```
String name = new String("Вася");  
name = new String(); // пустая строка
```

При первом использовании строкового литерала в программе, в памяти создается новый объект класса String и ссылка на него присваивается в переменную. Объект строки создается в специально отведенном месте памяти, называемом пул строк*. При повторном использовании литерала, новый объект не создается, а в переменную присваивается ссылка на ранее созданный объект.

```
String name1 = "Вася"; // создается объект новой строки и помещается в пул строк.  
String name2 = "Вася"; // в переменную присваивается ссылка на ранее созданный объект.
```

Статические методы класса String

■ **valueOf** – возвращает строковое представление значений примитивных типов.

```
String value = String.valueOf(3);  
value = String.valueOf(0.5);  
value = String.valueOf(true);
```

■ **format** – метод возвращает строку, в которой спецификаторы формата заменены на значения параметров в методе.

Синтаксис:

```
String.valueOf(<строка шаблона>, <подставляемые значения>);
```

```
String result = String.valueOf("Жили у бабули %d веселых гуся", 2);
```

При форматировании используются спецификаторы формата:

Специ- фикатор форма- та	Выполняемое форматирование
%a	Шестнадцатеричное значение с плавающей точкой
%b	Логическое (булево) значение аргумента
%c	Символьное представление аргумента
%d	Десятичное целое значение аргумента
%h	Хэш-код аргумента
%e	Экспоненциальное представление аргумента
%f	Десятичное значение с плавающей точкой
%g	Выбирает более короткое представление из двух: %e или %f
%o	Восьмеричное целое значение аргумента
%n	Вставка символа новой строки
%s	Строковое представление аргумента
%t	Время и дата
%x	Шестнадцатеричное целое значение аргумента
%%	Вставка знака %

Escape-последовательности

Некоторые символы нельзя вставлять в строку (например, двойные кавычки), для таких символов применяется экранирование с помощью косой черты \.

```
String text = "\"Символ, которому предшествует наклонная черта влево (\\), является  
escape-последовательностью.\"";
```

Escape- последовательность	Описание
\t	Символ табуляции
\b	Символ Backspace
\n	Новая строка
\r	Возврат каретки
\f	Перевод формата
\'	Символ одинарной кавычки
\"	Символ двойной кавычки
\\	Символ наклонной черты влево

Методы экземпляра

Все методы класса String не изменяют строку, у которой они вызываются, а возвращают ссылку на новый объект класса String.

- **charAt** – возвращает символ из строки, находящийся по индексу.
- **concat** – возвращает конкатенацию (объединение двух строк в одну).

```
String firstName= "Иван";  
String lastName= "Иванов";  
String fullname = cat.concat(name);  
// аналогично  
// firstName+ " " + lastName;  
System.out.println(fullname);
```

- **length** – возвращает количество символов в строке.

■ **isEmpty** – возвращает истину, если строка не содержит символов, иначе ложь. Работает быстрее, чем `length`.

■ **charAt** – возвращает символ из строки по индексу.

```
String testString = "test";  
char myChar = testString.charAt(3);  
System.out.println(myChar);
```

Результат: t

■ **contains** – возвращает истину, если строка содержит хотя бы одно совпадение со сравниваемой строкой.

```
String testString = "testing";  
boolean test = testString.contains("st");  
System.out.println(test);
```

Результат: true

■ **startsWith** – возвращает истину, если строка начинается с искомого символа или строки.

```
String str1 = "Star Wars";  
boolean test = str1.startsWith("Star");  
System.out.println(test);
```

Результат: true

■ **endsWith** – возвращает истину, если строка заканчивается на искомый символ или строку.

```
String str1 = "Java.exe";  
boolean test = str1.endsWith(".exe");  
System.out.println(test);
```

Результат: true

■ **trim** – возвращает строку с удаленными начальными и конечными пробелами.

■ **toLowerCase** – возвращает строку, в которой все заглавные символы исходной строки заменены на строчные.

■ **toUpperCase** – возвращает строку, в которой все строчные символы исходной строки заменены на заглавные.

■ **indexOf** – возвращает индекс символа, с которого найдено первое совпадение с искомой строкой или символом. Поиск начинается с начала строки. Если совпадение не найдено, возвращает `-1`.

```
String text = "оборонеспособность";  
int index = text.indexOf('б');  
System.out.println(index);
```

Результат: 1

Метод `indexOf` позволяет начать поиск не с начала, а с определенного символа.

```
String text = "оборонеспособность";
```

```
int index = text.indexOf('б');  
index = text.indexOf('б', index + 1);  
System.out.println(index);
```

Результат: 1

■ **lastIndexOf** – возвращает индекс символа, с которого найдено первое совпадение с искомой строкой или символом. Поиск начинается с конца строки. Если совпадение не найдено, возвращает -1.

```
int index = "readme.txt".lastIndexOf(".");  
System.out.println(index);
```

Результат: 6

■ **substring** – возвращает часть строки из исходной.

```
// вырезает строку с 6 символа и до конца строки.  
String world = "Hello World".substring(6);  
System.out.println(world);
```

Результат: World

```
String world = "Hello World".substring(1, 4);  
System.out.println(world);
```

Результат: ell

Примечание. Метод `substring` возвращает новую строку, но массив с символами берется из исходной строки, изменяется только индекс смещения с начала массива (`offset`) и количество элементов от индекса смещения (`count`), в связи с этим сборщик мусора не сможет очистить память от неиспользуемой части строки.

■ **replace** – возвращает строку, заменяя в исходной строке символ или набор символов на другой символ или набор символов.

■ **getBytes** – возвращает строку в виде массива байт.

Класс **StringBuffer**

Как уже упоминалось ранее, внутренние данные объекта класса `String` нельзя изменить после его создания, поэтому для работы со строками как с изменяемой структурой, применяется класс **StringBuffer**. В отличие от строк, объекты класса `StringBuffer` нельзя создавать с помощью строковых литералов, только с помощью оператора **new**.

```
StringBuffer sb = "test"; // ошибка компиляции  
StringBuffer sb = new StringBuffer("test");
```

Методы класса **StringBuffer**

■ **append** – добавляет новую строку в объект класса `StringBuffer` и возвращает ссылку на этот же объект.

```
StringBuffer sb = new StringBuffer("test");  
// создание объекта  
sb.append('-').append("test");  
// добавление значений цепочкой
```

```
sb.append(true); // единичное добавление
sb.append(1);
System.out.println(sb);
```

Результат: test-testtrue1

Примечание. Конкатенация строк в цикле объектов класса String с использованием оператора + может привести к проблемам с производительностью в связи с тем, что постоянно создаются новые объекты для строк. В такой ситуации предпочтительнее использовать объект класса StringBuffer или StringBuilder.

■ **insert** – вставляет строку или символ в объект класса StringBuffer.

```
StringBuffer sb = new StringBuffer("I Java!");
sb.insert(2, "love"); // 2 - индекс символа, после которого будет вставлена строка
System.out.println(sb.toString());
```

Результат: I love Java!

■ **reverse** – меняет порядок символов на обратный.

```
StringBuffer sb = new StringBuffer("palindrome");
sb.reverse();
System.out.println(sb);
```

Результат: emordnilap

■ **delete** – удаляет часть строки начиная с указанного индекса.

```
StringBuffer phrases = new StringBuffer("I do not like Java!");
phrases.delete(2, 7);
System.out.println(phrases);
```

■ **deleteCharAt** – удаляет символ из строки, по указанному индексу.

■ **replace** – заменяет подстроку в строке.

StringBuilder, его отличия от класса StringBuffer

Класс StringBuilder имеет точно такое же предназначение и такие же методы, как и StringBuffer. Класс StringBuilder отличается от StringBuffer отсутствием синхронизации данных для многопоточного программирования, поэтому его использование в однопоточной программе предпочтительнее.

Метод toString()

Все классы в Java неявно наследуются от класса Object. В классе Object имеется метод toString(), который возвращает строковое представление состояния объекта. В классах StringBuilder и StringBuffer этот метод возвращает объект класса String, который содержит строку, хранящуюся в объекте классов StringBuilder или StringBuffer на момент вызова метода.

Класс StringTokenizer

Класс StringTokenizer предназначен для разбиения строки на части (токены). В качестве разделителя используется регулярное выражение, переданное вторым параметром в конструктор класса.

Пример разбиения строки на слова, используя в качестве разделителя символ пробела:

```
String s = "Best Java programming language.";
StringTokenizer st = new StringTokenizer(s);
while (st.hasMoreTokens())
{
    System.out.println(st.nextToken());
}
```

Пример разбиения строки на слова, используя в качестве разделителя запятую:

```
String s = "Best,Java,programming,language.";
StringTokenizer st = new StringTokenizer(s, ",");
while (st.hasMoreTokens())
{
    System.out.println(st.nextToken());
}
```

Особенности конкатенации строк с другими типами

Строки, полученные в результате конкатенации строковых литералов и переменных, в пул строк не помещаются.

```
String t = "t";
String cat1 = "ca" + t;
String cat2 = "cat";
System.out.println(cat1 == cat2);
```

Результат: false

В случае конкатенации строк оператором + с другими примитивными типами данных, все последующие типы преобразуются в строчное представление. В связи с этим можно получить не то, что ожидается.

```
String text = "sum =" + 1 + 2; // 1 и 2 преобразуются в строку
System.out.println(text );
```

Результат: sum =12

Добавление круглых скобок в выражение меняют порядок выполнения. Сначала произойдет операция сложения литералов и только потом преобразование к строке.

Пример с скобками:

```
String text = "sum =" + (1 + 2);
System.out.println(text );
```

Результат: sum =3

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

Результат: 3 = 3

Пример с комбинацией операторов:

```
String text = "mul =" + 3 * 2;  
System.out.println(text );
```

Результат: mul =6

Сравнение строк

При применении оператора сравнения (==) для переменных ссылочного типа происходит сравнение ссылок на объект, хранящихся в переменных. В связи с этим, если создать две строки с идентичным содержанием, через оператор new, сравнение таких строк оператором == вернет false. В связи с этим для сравнение идентичности разных строк, следует использовать метод equals, который сравнивает строки на эквивалентность.

Пример сравнения строк, инициализированных литералами:

```
String text1 = "Java";  
String text2 = "Java"; // в переменную присваивается ссылка на объект из пула строк  
System.out.println(text1 == text2);  
System.out.println(text1.equals(text2));
```

Результат: true, true

Пример сравнения строк, созданных через оператор new:

```
String text1 = new String("Java");  
String text2 = new String("Java");  
System.out.println(text1 == text2);  
System.out.println(text1.equals(text2));
```

Результат: false, true

Строку, созданную с помощью конкатенации или оператора new, можно программно поместить в "пул строк", используя **метод intern()**. Метод intern() проверяет, есть ли данная строка в пуле строк, и если такая строка отсутствует, помещает ее в пул строк. Возвращает ссылку на строку из пула строк.

```
String text1 = new String("Java");  
text1 = text1.intern();  
text2 = text2.intern();  
String text2 = new String("Java");  
System.out.println(text1 == text2);  
System.out.println(text1.equals(text2));
```

Результат: true, true

Важно!!! В классах StringBuilder и StringBuffer метод equals не переопределен и, следуя стандартной реализации в классе Object, сравнивает ссылки на объекты. В связи с этим при необходимости сравнить два объекта классов StringBuilder или StringBuffer, преобразуйте оба объекта к строке методом toString.

```
StringBuffer sb1 = new StringBuffer("abc");  
StringBuffer sb2 = new StringBuffer("abc");  
System.out.println(sb1.equals(sb2)); // неверно
```

Результат: false

Регулярные выражения

Регулярные выражения – это формальный язык, предназначенный для поиска и различных манипуляций с подстроками текста. Основой языка являются метасимволы. Регулярная строка является образцом или шаблоном, который задает правила поиска подстрок в исходной строке.

Регулярные выражения являются мощным и гибким инструментом для работы с текстом, в разы уменьшают количество исходного кода, но использование регулярных выражений часто усложняет понимание и чтение кода. Поэтому рекомендуется очень тщательно продумывать необходимость использования регулярных выражений.

Большинство символов в регулярных выражениях представляют самих себя, кроме метасимволов.

Основные метасимволы регулярных выражений:

Мета-символ	Назначение
^	Начало проверяемой строки
\$	Конец проверяемой строки
.	Сокращенная форма записи для символьного класса, совпадающего с любым символом
	Подвыражения, объединенные этим способом, называются альтернативами (alternatives)
?	Предшествующий ему символ является необязательным
+	Обозначает «один или несколько экземпляров непосредственно предшествующего элемента»
*	Любое количество экземпляров элемента (в том числе и нулевое)
\\d	Цифровой символ
\\D	Нецифровой символ
\\s	Пробельный символ
\\S	Любой символ, кроме пробела
\\W	Любой символ, кроме буквенного или цифрового символа или знака подчёркивания
\\w	Буквенный или цифровой символ или знак подчёркивания

В классе String есть несколько методов использующих регулярные выражения:

■ **matches** – метод возвращает истину, если исходная строка соответствует регулярному выражению, иначе возвращает ложь.

Пример проверяет, соответствует ли строка правилам составления электронного адреса:

```
String email = "unguryan@itstep.org";
```



```
// регулярное выражение для проверки правильности email адреса
String regular = "[a-zA-Z]{1}[a-zA-Z\\d\\u002E\\u005F]+@[a-zA-Z]+\\u002E{1,2}((net)|(com)|(org))";
System.out.println(email.matches(regular));
```

Результат: true

■ **replaceAll** – возвращает строку, в которой все подстроки, удовлетворяющие условию регулярного выражения, заменены на строку из второго параметра.

```
String text= "I love coffee!";
text = text.replaceAll("[Cc]offee", "Java");
```

■ **split** – возвращает массив строк, разбивая строку на части, используя в качестве разделителя регулярное выражение. В Java есть пакет `Java.util.regex`, предназначенный для работы с регулярными выражениями.

```
String url= "http://mystat.itstep.org/";
// регулярное выражение для проверки правильности URL адреса
String regular = "/^((?:[a-z]+):(?:[a-z]*)?)\\W)?(?:([^\@]*)?@)?((?:[a-z0-9_-]+\\.)+[a-z]{2,}|localhost|(?:(?:[01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.){3}(?:[01]?\\d\\d?|2[0-4]\\d|25[0-5]))(?:\\d+)?(?:[^\?\\#]+)?(?:\\?\\?([^\#]+)?(?:\\#([^\s]+))?$|/)"
Pattern p = Pattern.compile(regular); // создает объект образец
Matcher m = p.matcher(url); // создает объект совпадений
System.out.println(m.matches());
```

Результат: true

Сравнение ссылок

<https://javarush.com/>

Присваивание ссылок на строки. Выгода такого подхода становится очевидной, если вам нужно присвоить одной строковой переменной другую строковую переменную

```
String text = "Это очень важное сообщение";
String message = text;
```

Работа со ссылками и объектами

```
String text = "Это очень важное сообщение";
String message = text.toUpperCase();
toUpperCase() не меняет ту строку, у которой он был вызван. Вместо этого он создает новую строку (новый объект) и возвращает ссылку на него.
```

Скажем, вы решили передать строку в объект типа `Scanner` (чтобы он читал значения из нее).

```
String text = "10 20 40 80";
Scanner console = new Scanner(text);
int a = console.nextInt();
int b = console.nextInt();
```

Сравнение ссылок на объекты типа String

Для сравнения строковых переменных можно использовать два оператора: == (равно) и != (не равно). Операторы «больше», «меньше», «больше либо равно» использовать нельзя — компилятор не допустит. Но есть интересный нюанс: что у нас хранится в строковых переменных? Правильно: адреса (ссылки) на объекты. Вот эти самые адреса сравниваться и будут:

```
String text = "Привет";
String message = text;
String s1 = text.toUpperCase();
String s2 = text.toUpperCase();
```

Сравнение строк по содержанию

Чтобы программа при сравнении строк смотрела не на адреса объектов String, а на их содержимое, есть специальный метод — equals.

строка1.equals(строка2)

Этот метод возвращает true (истина), если строки одинаковые, и false (ложь), если они не одинаковые.

```
String s1 = "Привет";
String s2 = "ПРИВЕТ";
String s3 = s1.toUpperCase();

System.out.println(s1.equals(s2));
System.out.println(s1.equals(s3));
System.out.println(s2.equals(s3));
```

Напиши программу, которая считывает с клавиатуры две строки и выдает сообщение о том, одинаковые ли эти строки.

```
public class Solution {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String s1 = scanner.nextLine();
        String s2 = scanner.nextLine();
        if (s1.equals(s2))
            System.out.println("строки одинаковые");
        else
            System.out.println("строки разные");
    }
}
```

В методе main напиши код попарного сравнения по ссылке строк string1, string2 и string3 с выводом соответствующего сообщения после каждого сравнения:

"ссылки на строки одинаковые" или "ссылки на строки разные".

Порядок сравнения должен быть следующим:

- строку string1 со строкой string2
- строку string2 со строкой string3
- строку string1 со строкой string3.

```
public class Solution {
    public static String string1 = "Амиго";
```

```

public static String string2 = string1;
public static String string3 = new String(string1);

public static void main(String[] args) {
    String same = "ссылки на строки одинаковые";
    String different = "ссылки на строки разные";
    System.out.println(string1 == string2 ? same : different);
    System.out.println(string2 == string3 ? same : different);
    System.out.println(string1 == string3 ? same : different);
}
}

```

Сравнение строк без учета регистра

есть еще один специальный метод — `equalsIgnoreCase`. Выглядит его вызов так:
`строка1.equalsIgnoreCase(строка2)`

```

String s1 = "Привет";
String s2 = "ПРИВЕТ";
String s3 = s1.toUpperCase();
System.out.println(s1.equalsIgnoreCase(s2));
System.out.println(s1.equalsIgnoreCase(s3));
System.out.println(s2.equalsIgnoreCase(s3));

```

Амиго создал секретное слово для доступа к закрытой информации, но у него сломалась клавиатура, и теперь невозможно набирать буквы в верхнем регистре. Напиши программу, которая будет сравнивать введенную строку со строкой из переменной `secret`, не учитывая регистр. Если введенная строка равна строке из переменной `secret`, программа выводит на экран сообщение "доступ разрешен". В ином случае - "доступ запрещен".

```

public class Solution {
    public static String secret = "AmIgo";
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String s1 = scanner.nextLine();
        String result = secret.equalsIgnoreCase(s1) ? "доступ разрешен" : "доступ запрещен";
        System.out.println(result);
        // if(s1.equalsIgnoreCase(secret));
        // System.out.println("доступ разрешен");
        // else
        // System.out.println("доступ запрещен");
    }
}

```

Пример сравнения строк

```

Scanner console = new Scanner(System.in);
String a = console.nextLine();
String b = console.nextLine();
String result = a.equals(b) ? "Одинаковые" : "Разные";
System.out.println(result);

```

Методы

<https://www.sololearn.com/>

Метод - это блок кода, предназначенный для выполнения определенной задачи. `Println()`, который мы используем для вывода, также является методом. Цель метода состоит в том, чтобы создать его один раз и вызывать несколько раз, когда это необходимо для выполнения определенных задач.

Статический необходим для того, чтобы иметь возможность использовать метод в `main`. `void` означает, что этот метод не имеет возвращаемого значения. `добро пожаловать` - это название метода.

```
static void welcome() {  
    System.out.println("Welcome");  
    System.out.println("I am a method");  
    System.out.println("End of method");  
}
```

Чтобы вызвать метод, введите его имя, за которым следует набор круглых скобок.

```
public class Demo {  
    static void welcome() {  
        System.out.println("Welcome");  
        System.out.println("I am a method");  
        System.out.println("End of method");  
    }  
    public static void main(String[] args) {  
        welcome();  
    }  
}
```

Вы создаете программу автоматического реагирования для магазина. Бот должен принять номер от пользователя в качестве входных данных и ответить автоматическим сообщением. В настоящее время есть 3 ответа, которые вам нужны для обработки: Сообщение пользователя: "1", ответ: "Заказ подтвержден" Сообщение пользователя: "2", ответ: "info@sololearn.com " Для любого другого номера ответ должен быть таким: "Попробуйте еще раз".

```

import java.util.Scanner;
public class Program {

    static void bot() {
        Scanner console = new Scanner(System.in);
        int a = console.nextInt();

        if (a==1) {
            System.out.println("Order confirmed");
        }
        else if (a==2) {
            System.out.println("info@sololearn.com");
        }
        else {
            System.out.println("Try again");
        }
    }
    public static void main(String[] args) {
        bot();
    }
}

```

Параметры метода (Method Parameters)

```

static void welcome(String name) {
    System.out.println("Welcome, " + name);
}

```

Описанный выше метод принимает в качестве параметра строку с именем name, которая используется в методе.

```

class Demo {
    static void welcome(String name) {
        System.out.println("Welcome, "+name);
    }
    public static void main(String[] args) {
        welcome("James");
        welcome("Amy");
    }
}

```

Значения, передаваемые в качестве параметров, называются аргументами.

Заполните пробелы, чтобы определить метод doubleNum, который принимает целое число в качестве параметра и выводит его значение double. Затем вызовите его в main с аргументом 42.

```

static void doubleNum (int num) {
    System.out.println("num*2);
}
public static void main(String[] args) {
    doubleNum (42)
}

```

Множество параметров

Методы могут принимать несколько параметров. Для этого нам просто нужно разделить их запятыми, например: Теперь наш метод `welcome()` принимает строку и целое число в качестве своих параметров.

```
static void welcome(String name, int age) {  
    System.out.println("Welcome, "+name);  
    System.out.println("Your age: "+age);  
}
```

Теперь, при вызове функции, нам нужно указать все параметры

```
class Demo {  
    static void welcome(String name, int age) {  
        System.out.println("Welcome, "+name);  
        System.out.println("Your age: "+age);  
    }  
    public static void main(String[] args) {  
        welcome("James", 42);  
        welcome("Amy", 25);  
    }  
}
```

Заполните пробелы, чтобы определить метод, который принимает два целых числа и выводит их сумму. Затем вызовите его в `main` для аргументов `x` и `y`.

```
static void sum (int a, int b) {  
    System.out.println(a+b);  
}  
public static void main(String[] args) {  
    int x= 8;  
    int y= 11;  
    sum (x, y)  
}
```

Параметры метода действительно удобны!

Вот краткое изложение:

- Вы можете определить параметры в круглых скобках.
- Несколько параметров должны быть разделены запятыми.
- Параметры доступны в методе, как и переменные с заданными именами.
- При вызове метода вам необходимо указать его параметры в том же порядке, в каком они определены.

Например, мы можем создать метод для вычисления заданного процента от числа и вывести его:

```
class Demo {  
    static void perc(double num, int percentage) {  
        double res = num*percentage/100;  
        System.out.println(res);  
    }  
    public static void main(String[] args) {  
        perc(530, 23);  
    }  
}
```

What is the result of `demo(8, 3)`? ответ $8/3=2$

```

static void demo(int x, int y) {
    if(x<y) {
        System.out.println(x+y);
    } else {
        System.out.println(x%y);
    }
}
}
}
}

```

Преобразователь футов в дюймы. Вам нужно создать метод, который преобразует значение фута в дюймы. В 1 футе содержится 12 дюймов. Определите метод convert(), который принимает значение foot в качестве аргумента и выводит значение inches. Результат должен быть двойным. Данный код принимает значение foot в качестве входных данных и передает его методу convert. Определите метод преобразования, чтобы данный код работал.

```

import java.util.Scanner;
public class Program {
    static void convert(double num) {
        int feet = (int) (num * 12);
        Double d = Double.valueOf(feet);
        System.out.println(d);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double num = sc.nextDouble();
        convert(num);
    }
}
или

static void convert(double num) {
    System.out.println(num*12);
}
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    double num = sc.nextDouble();
    convert(num);
}
}

```

Определить метод с двумя параметрами, называемыми area. Он должен вычислить и вывести площадь прямоугольника на основе ширины и высоты.

```

static void area (double w, double h) {
    double res = w*h;
    System.out.println(res);
}

```

Returning From Methods (Возврат из методов)

Вот такое же определение функции, указывающее, что возвращаемый тип должен быть двойным:

```

static double perc(double num, int percentage) {
    ...
}

```

```
}
```

Это означает, что наш метод `perc` вернет значение типа `double`.

Теперь мы можем вернуть наш результат, используя *ключевое слово* `return`: Ключевое слово `return` останавливает выполнение метода. Если после `return` будут какие-либо инструкции, они не будут выполняться.

```
static double perc(double num, int percentage) {  
    double res = num*percentage/100;  
    return res;  
}
```

После того как мы создали наш метод, возвращающий значение, мы можем вызвать его в нашем коде и присвоить результат переменной. Returning is useful when you don't need to print the result of the method, but need to use it in your code. For example, a bank account's `withdraw()` method could return the remaining balance of the account

```
double x = perc(530, 23);  
System.out.println("Result is: "+x);
```

Заполните пробелы, чтобы создать метод, который принимает два целых числа и возвращает их сумму. Затем вызовите его в `main`.

```
static int sum(int x, int y) {  
    int res = x+y;  
    return res;  
}  
public static void main(String[] args) {  
    int n1 = 33;  
    int n2 = 12;  
    int n = sum (n1, n2);  
}
```

Возвращаемое значение.

Давайте создадим метод, который принимает два параметра, `string name` и оценку типа `integer`, проверяет, превышает ли оценка 70, и возвращает `boolean` результат. Затем давайте используем его в `main`. Как вы можете видеть, мы можем использовать метод в операторе `if`, поскольку он возвращает логическое значение.

```
class Demo {  
    static boolean check(int grade) {  
        if(grade >=70) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
    public static void main(String[] args) {  
        int x = 89;  
        if(check(x) == true) {  
            System.out.println("Congrats!");  
        }  
    }  
}
```


Вы создаете конвертер градусов Цельсия в градусы Фаренгейта. Напишите метод, который принимает значение по Цельсию в качестве аргумента и возвращает соответствующее значение по Фаренгейту. Данный код принимает значение по Цельсию в качестве входных данных и передает его методу `fahr()`, который вам нужно создать. Для вычисления значения по Фаренгейту используется следующее уравнение: $1,8 * \text{по Цельсию} + 32$.

```
import java.util.Scanner;
public class Program {
    static double fahr(double celsius) {
        return(1.8 * celsius + 32);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double c = sc.nextDouble();

        System.out.println(fahr(c));
    }
}
```

Заполните пробелы, чтобы создать метод с именем `checkAge`, который возвращает значение `true`, если его параметр `age` больше 18.

```
static boolean checkAge(int age) {
    if (age >= 18) {
        return true;
    } else {
        return false;
    }
}
```

чтобы определить метод, который не возвращает значение.

```
static void printBill(double amount) { //some code
}
```

Измените порядок кода, чтобы объявить метод, который возвращает квадрат своего аргумента.

```
public int max (int a) {
    int result=a*a;
    return result }
}
```

Объявить метод, который принимает один аргумент типа `int` и возвращает факториал этого числа. Факториал - это произведение всех целых чисел от 1 до заданного числа.

```
static int fact( int num) {
    int res = 1;
    for(int x=1; x <= num; x++) {
        res *= x;
    }
    return res;
}
```

Создать метод под названием `check`, который принимает два параметра: имя и возраст. Выведите имя, если возраст больше 17 лет, и ошибку, если это не так.

```
static void check (string name, int age) {
```

```
if (age > 17) {  
    System.out.println(name);  
} else {  
    System.out.println("Error");  
}  
}
```

Введение в классы, объекты и методы 136 стр (руководство для начинающих 9)
Объявление импорта Java_для_чайников_7_е_издание_Барри_Берд стр 111
ПРИМИТИВНЫЕ ТИПЫ И ССЫЛОЧНЫЕ ТИПЫ стр 62 java быстрый старт

<http://www.softelectro.ru/ieee754.html> **ЧИТАТЬ**

<https://www.youtube.com/embed/64E3t7I3iKw> **СМОТРЕТЬ**

https://developer.alexanderklimov.ru/android/java/logic_operators.php **ЧИТАТЬ**

<http://study-java.ru/uroki-java/urok-8-java-logicheskyye-i-uslovnyye-operatory/> **ЧИТАТЬ**

<https://dark-barker.blogspot.com/2012/03/bit-operations-java-pitfalls.html> **ЧИТАТЬ**

<http://study-java.ru/uroki-java/java-operatory-tsikla-for-while-do-while-operator-break/> **ЧИТАТЬ**

<https://habr.com/ru/articles/174065/> **ЧИТАТЬ**

<https://metanit.com/java/tutorial/2.9.php> **ЧИТАТЬ**

<https://cdn.cs50.net/2016/x/psets/0/pset0/bulbs.html> **ЧИТАТЬ**

https://www.tutorialspoint.com/java/java_for_loop **ЧИТАТЬ**

https://www.tutorialspoint.com/java/java_while_loop **ЧИТАТЬ**

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html> **ЧИТАТЬ**

Спецификация виртуальной машины Java™ Второе издание **ЧИТАТЬ**

<https://docs.oracle.com/javase/specs/jvms/se6/html/VMSpecTOC.doc.html>

https://www.tutorialspoint.com/java/java_basic_operators **ЧИТАТЬ**

https://www.tutorialspoint.com/java/java_basic_datatypes **ЧИТАТЬ**

<https://www.youtube.com/watch?v=6pApP6TKxck> **СЛУШАТЬ**

Урок 2 - Синтаксис языка - Java для тестировщиков

<https://www.youtube.com/watch?v=x82BOGrO310> **СЛУШАТЬ**

Уроки Java - №3 Прimitives типы данных и переменные

https://www.youtube.com/watch?v=SW_UCzFO7X0 **СЛУШАТЬ**

CS50 на русском: Лекция #1 [Гарвард, Основы программирования, осень 2015 год]

<https://www.youtube.com/watch?v=grEKMHGyYns> **СЛУШАТЬ**

Learn Java 8 - Full Tutorial for Beginners

Из видео <https://journal.top-academy.ru/ru/main/library/page/index/5>

<https://www.youtube.com/watch?v=DKy4KzIn1qg&t=1s>