**Программирование на Джава**

**Список вопросов к экзамену**

**Вопросов 0 / 95**

**doc: \***

**Статусы вопросов:**

**👍 - Вопрос выполнен и проверен (Только администраторы)**

**✏️ - Требует правки**

**🔒 - На проверке, не изменять**

**❤️ - Вопрос выполнен, но не проверен**

**🧡 - Сделаны почти все пункты, но нужно дополнить**

**💚 - Сделана половина вопроса**

**💙 - Вопрос начат**

**🤍 - Вопрос занят**

**Если вы ставите или изменяете вопрос, то сразу под вопросом, вставьте ссылку на вк (если дополняете вопрос, ссылку через запятую)**

**Тема 1. Особенности платформы Java. Синтаксис языка Java**

**1. 🔒 К какому типу языков относится язык Джава**

Java — язык программирования общего назначения. Относится к объектно-ориентированным языкам программирования, к языкам с сильной типизацией. Создатели реализовали принцип WORA: write once, run anywhere или «пиши один раз, запускай везде».

**2. 🔒 Особенности языка Джава**

Создатели реализовали принцип WORA: write once, run anywhere или «пиши один раз, запускай везде». Это значит, что написанное на Java приложение можно запустить на любой платформе, если на ней установлена среда исполнения Java (JRE, Java Runtime Environment).

Эта задача решается благодаря компиляции написанного на Java кода в байт-код. А затем этот байт-код выполняется виртуальной машиной JVM (Java Virtual Machine).

В Java реализован механизм управления памятью, который называется сборщиком мусора или garbage collector. Разработчик создаёт объекты, а JRE с помощью сборщика мусора очищает память, когда объекты перестают использоваться. «Есть такое понятие — циклический мусор. Внутри цикла на все объекты есть ссылки, однако garbage collector в Java удалит его, если объекты не могут использоваться из программы».

Синтаксис языка Java похож на синтаксис других си-подобных языков. Вот его некоторые особенности:

● чувствительность к регистру — идентификаторы User и user в Java представляют собой разные сущности;

● для именования методов используется lo  werCamelCase. Если название метода состоит из одного слова, оно должно начинаться со строчной буквы. Пример: firstMethodName();

● для именования классов используется UpperCamelCase. Если название состоит из одного слова, оно должно начинаться с прописной буквы. Пример: FirstClassName.

● название файлов программы должно точно совпадать с названием класса с учётом чувствительности к регистру. Например, если класс называется FirstClassName, файл должен называться FirstClassName.java;

● идентификаторы всегда начинаются с буквы (A-Z, a-z), знака $ или нижнего подчёркивания \_;

**3. 🔒 Класс Scanner и его использование для чтения стандартного потока ввода**

@sanya.bog.batya

Для ввода данных используется класс Scanner из пакета java.util.

Этот класс надо импортировать в ту программу, где он будет использоваться. Это делается до начала определения класса в коде программы.

В классе есть методы для чтения очередного символа заданного типа со стандартного потока ввода, а также для проверки существования такого символа.

Для работы с потоком ввода необходимо создать объект класса Scanner, при создании указав, с каким потоком ввода он будет связан. Стандартный поток ввода (клавиатура) в Джава представлен объектом — System.in.

//Импортируем Scanner

import java.util.Scanner;

public class Main {

   public static void main(String[] args) {

       //Создаем объект класса сканер

       Scanner nameOfScanner = new Scanner(System.*in*);

       //Пример работы со сканером

       String nameOfString;

       nameOfString = nameOfScanner.nextLine();

       int nameOfInteger;

       nameOfInteger = nameOfScanner.nextInt();

   }

}

**4. 🔒 Класс Scanner, конструктор класса Scanner для чтения стандартного потока ввода**

@sanya.bog.batya

Конструктор класса сканер создает новый сканер, который в свою очередь создает значения, отсканированные из указанного источника. В параметры конструктора и вписывается источник.

**public Scanner(Readable source):**

Создает новый сканер, который создает значения, отсканированные из указанного источника.

Параметры: source – источник символов, реализующий интерфейс Readable

FileReader reader = **new** FileReader(**new** File("D:\temp\test.txt"));

       Scanner scan = **new** Scanner(reader);

**public Scanner(InputStream source):**

Создает новый сканер, который создает значения, отсканированные из указанного входного потока. Байты из потока преобразуются в символы с использованием кодировки по умолчанию базовой платформы.

Параметры: источник – входной поток для сканирования.

       File input = **new** File("D:\temp\test.txt");

      FileInputStream fis = **new** FileInputStream(input);

     Scanner scan = **new** Scanner(fis);

**public Scanner(File source) выдает исключение FileNotFoundException:**

Байты из файла преобразуются в символы с кодировкой по умолчанию базовой платформы.

Параметры: источник – файл для сканирования

Этот конструктор очень прост. Просто требует источник файла. Единственной целью этого конструктора является создание экземпляра объекта Scanner для сканирования через файл.

File input = **new** File("D:\temp\test.txt");

       Scanner scanner = **new** Scanner(input);

**public Scanner(Path source) throws IOException:**

источник – путь к файлу для сканирования. Для параметра конструктора требуется источник Path, который используется редко.

      Path path = FileSystems.getDefault().getPath("access.log");

      System.out.println(path.toUri());

     Scanner scannner = **new** Scanner(path);

**public Scanner(String source):**

Создает новый сканер, который выдает значения, отсканированные из указанной строки.

Источник – строка для сканирования.

Этот конструктор может быть самым простым на практике, поскольку для него требуется только строковый параметр, и он строго используется для сканирования ввода строки.

**String** source = "This is a test";

       Scanner scannner = **new** Scanner(source);

**5. 🔒 Методы класса Scanner nextLine(), nextInt(), hasNextInt(), hasNextLine() и их использование для чтения ввода пользователя с клавиатуры**

Для работы методов класса Scanner требуется подключить соответствующий класс.

https://lh3.googleusercontent.com/K7H9kcMh0bSk3XeDL3K-GhIjvdsTW8SibEqSMaWlALSuEUErgFuBMmJah8TiQ3O6BmBSvUng2oxrkVQQdwGCBqyBDUN3fbM56wouLBh0wBi9EJj-KFFT5IbNDDiP3ML4a80tXDgYZE91aWM_o56qQpK-8wBoWHXa1ZbPDOb6U1_ypAHAHQ4SDDHd1-5RrA

Метод **nextLine()** обращается к источнику данных, находит там следующую строку, которую он еще не считывал и возвращает ее.

Метод **nextInt()** считывает и возвращает введенное число.

Метод **hasNextInt()** проверяет, являются ли следующие данные числом или нет. Возвращается true или false.

Метод **hashNextLine()** проверяет, является ли следующие данные строкой.

<https://vk.com/dny_vin>

**6. 🔒 Примитивные типы данных, объявление и присваивание переменных**

<https://vk.com/id198165768>

В Java существует 8 примитивных типов данных:

● byte (целые числа, 1 байт) от -128 до 127

● short (целые числа, 2 байта) от -32768 до 32767

● int (целые числа, 4 байта) от -2147483648 до 2147483647

● long (целые числа, 8 байтов) от -9223372036854775808 до 9223372036854775807

● float (вещественные числа, 4 байта) от ~1,4\*10-45 до ~3,4\*1038

● double (вещественные числа, 8 байтов) от ~4,9\*10-324  до ~1,8\*10308

● char (символ Unicode, 2 байта) минимальное значение ‘\ u0000’ (или 0), и максимальное значение ‘\ uffff’ (или 65535 включительно).

● boolean (значение истина/ложь, 1 бит) true или false

Эти 8 типов служат основой для всех остальных типов данных. Примитивные

типы обладают явным диапазоном допустимых значений.

Объявление

 byte getByte, putByte;

short employeeID;

int i;

long l;

float f;

double d;

char c;

boolean bool;

Инициализация

 getByte = 0;

 shortCount = 0;

 intCount = 2147483647;

 floatCount = 44.03f;

doublePi = 3.14159;

charSim ='Ы';

booleanValue = true;

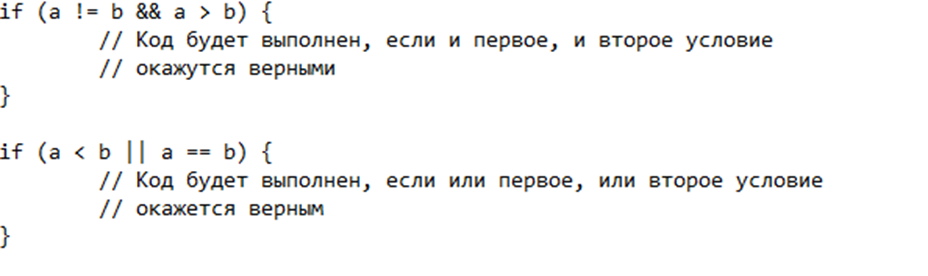
**7. 🔒 Условные операторы, полное и неполное ветвление в Джава, синтаксис**

Условные операторы языка if, if-else, switch. Само по себе ветвление – это алгоритмическая конструкция, в которой в зависимости от истинности некоторого условия, выполняется одна из нескольких последовательностей действий.

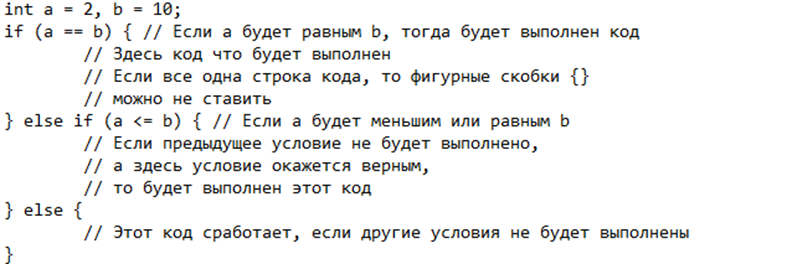
Полная форма ветвления: если <условие>, то <действие 1>, иначе <действие 2>. При полной форме ветвления действия выполняется в обоих случаях: и при истинности и при ложности условия.

Неполная форма ветвления: если <условие>, то <действие>, при нем действие отсутствует, если условие не выполняется.

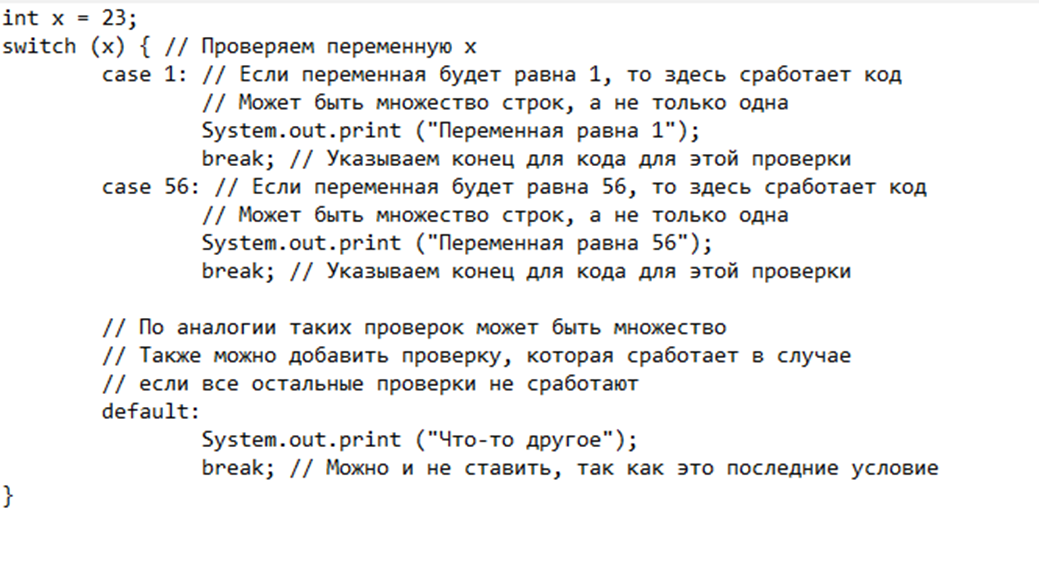
Пример конструкции if:



Пример конструкции if-else:

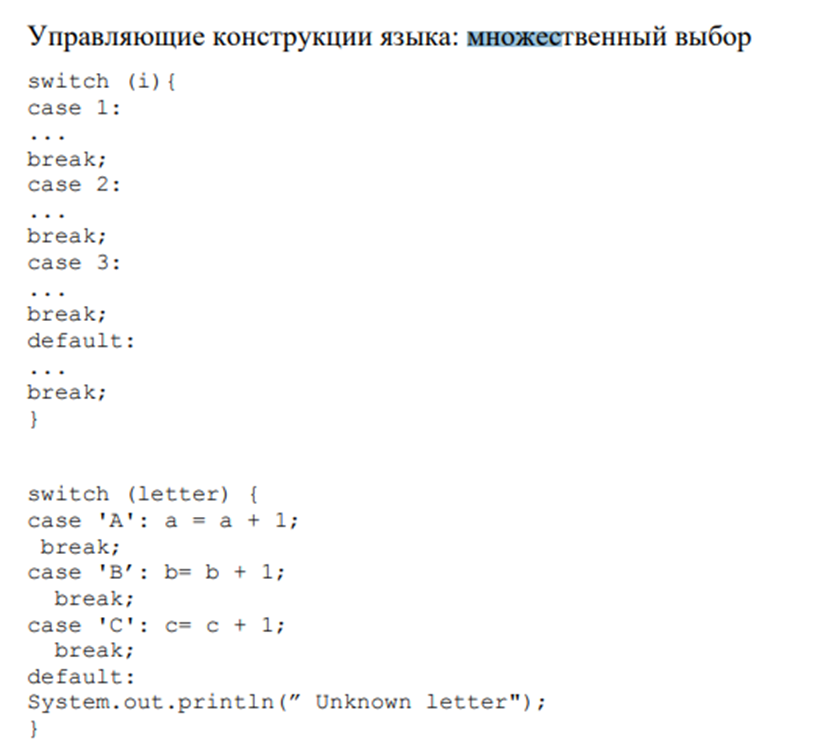


Пример конструкции switch:



**8. 🔒 Оператор множественного выбора в Джава, синтаксис**

Инструкция множественного выбора switch позволяет выполнять различные части программы в зависимости от того, какое значение будет иметь переменная (её называют «переменной-переключателем», а «switch» с английского переводится как раз как «переключатель»).



**9.  🔒 Класс System. Работа со стандартами потоками вывода**

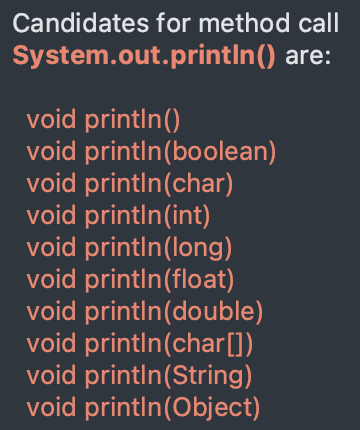
Класс System является финальным и не предоставляет общедоступных конструкторов. Из-за этого все члены и методы, содержащиеся в этом классе, являются статическими по природе.

Вызов метода из класса java.lang.System выглядит так: Команда System.out.print() выводит текст без пропуска строки, а команда System.out.println() выводит текст и ставит пропуск строки в конце.

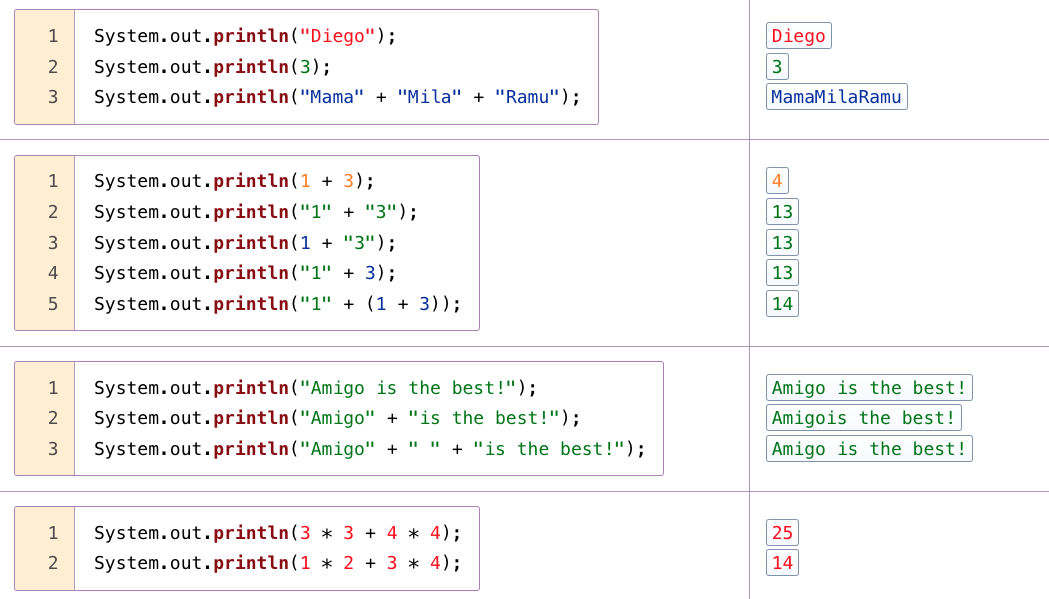
**10. 🔒 Перегруженные методы out.println() класса System и их использование для вывода в консоль**

<https://vk.com/l1senok>

Метод класса System out.println() перегружен для различных примитивов, String и Object, основан на методе toString():



Примеры использования:



**11. 🔒 Константы в Джава: объявление константы**

<https://vk.com/id198165768>

В отличие от переменных константам можно присвоить значение только один раз. Константа объявляется также, как и переменная, только вначале идет ключевое слово final:

final int LIMIT = 5;

System.out.println(LIMIT);  // 5

// LIMIT=57; // так мы уже не можем написать, так как LIMIT - константа

Как правило, константы имеют имена в верхнем регистре.

Константы позволяют задать такие переменные, которые не должны больше изменяться. Например, если у нас есть переменная для хранения числа pi, то мы можем объявить ее константой, так как её значение постоянно.

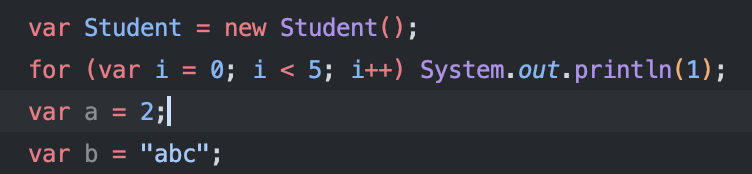
**12. 🔒 В результате выполнения этой строчки**

**13. 🔒 Объявление и использование бестиповых переменных в Джава**

<https://vk.com/l1senok>

В некоторых случаях, например когда тип переменной очевиден, разумнее инициализировать переменные ключевым словом **var**, однако присвоить им значение нужно сразу же.

Примеры:



**14.🔒 Объявление переменных и инициализация типа класс**

<https://vk.com/fanlug>

1. Объявление переменных

Объявление переменных в коде программы имеет вид:

int a;

или

float f  = 5.01f;

В первом случае это простое объявление, во втором случае - объявление и одновременная инициализация с определённым значением.

2.Инициализация типа класс

Инициализация типа класс в коде программы имеет вид:

Student student = new Student();

или

new Student();

или

Student student;

student = new Student();

В первом случае мы работаем с переменной ссылочного типа данных, которая содержит в себе ссылку на объект класса, который мы инициализировали, во втором случае - мы не создаём отдельной переменной - ссылки на объект класса, однако объект всё ещё находиться в системе. А в третьем объявили ссылочную переменную, и потом уже присвоили ей ссылку на инициализированный объект.

**15. 🔒 Арифметические операции, операции инкремента и декремента в Джава**

Операции в Java бывают унарные (выполняются над одним операндом), бинарные (над двумя операндами).

1. Операция сложения двух чисел

int a = 10;

int b = 7 ;

int c = a + b;

2. Операция вычитания двух чисел

int a = 10;

int b = 7;

int c = a - b;

3. Операция умножения двух чисел

int a = 10;

int b = 7;

int c = a \* b;

4. Операция деления двух чисел

int a = 10;

int b = 7;

int c = a / b;

При делении стоит учитывать, что если в операции участвуют два целых числа, то результат деления будет округляться до целого числа, даже если результат присваивается переменной float или double. Чтобы результат представлял число с плавающей точкой, один из операндов также должен представлять число с плавающей точкой.

5. Получение остатка от деления двух чисел:

int a = 33;

int b = 5;

int c = a % b; // 3

6. Инкремент

6.1 Префиксный инкремент

Предполагает увеличение переменной на единицу, например, z=++y (вначале значение переменной y увеличивается на 1, а затем ее значение присваивается переменной z);

int a = 8;

int b = ++a;

System.out.println(a); // 9

System.out.println(b); // 9

6.2 Постфиксный инкремент

Также представляет увеличение переменной на единицу, например, z=y++ (вначале значение переменной y присваивается переменной z, а потом значение переменной y увеличивается на 1)

int a = 8;

int b = a++;

System.out.println(a); // 9

System.out.println(b); // 8

7. Декремент

7.1 Префиксный декремент

Уменьшение переменной на единицу, например, z=--y (вначале значение переменной y уменьшается на 1, а потом ее значение присваивается переменной z)

int a = 8;

int b = --a;

System.out.println(a); // 7

System.out.println(b); // 7

7.2 Постфиксный декремент

z=y-- (сначала значение переменной y присваивается переменной z, а затем значение переменной y уменьшается на 1)

int a = 8;

int b = a--;

System.out.println(a); // 7

System.out.println(b); // 8

**16. 🔒 В результате выполнения фрагмента программы**

**17. 🔒 Арифметические операции, приоритет выполнения операций**

<https://vk.com/id198165768>

В Java есть бинарные арифметические операции и унарные . К бинарным операциям относят следующие:

* + операция сложения двух чисел
* - операция вычитания двух чисел
* \* операция умножения двух чисел
* / операция деления двух чисел
* % получение остатка от деления двух чисел

Также есть две унарные арифметические операции, которые производятся над одним числом: ++ (инкремент) и -- (декремент). Каждая из операций имеет две разновидности: префиксная и постфиксная:

**Операции в порядке уменьшения приоритета:**

++ (постфиксный инкремент), -- (постфиксный декремент)

++ (префиксный инкремент), -- (префиксный декремент)

\* (умножение), / (деление), % (остаток от деления)

+ (сложение), - (вычитание)

**18. ❤️ Типы данных в языке Джава, классификация, примеры**

Примитивные типы данных в языке Джава описаны в вопросе 6.

Также в языке Джава существуют ссылочные типы данных, это типы классов, интерфейсов и типы массивов, также существует специальный null тип.

Существует четыре вида ссылочных типов данных: классы, интерфейсы, ссылки на примитивные типы и массивы.

Примером классам могут быть строки - это объекты класса **String**.

Если требуется создать ссылку на примитивный тип данных,то необходимо использоваться класс-обертку.

|  |  |
| --- | --- |
| **Тип** | **Класс-обертка** |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| char | Character |
| float | Float |
| double | Double |
| boolean | Boolean |

Пример использования:

int i;

Integer obertka;

obertka = i;

Ссылочные типы данных создаются самим программистом, за исключением String и Array.

Ссылочные переменные хранят адрес ячейки памяти,в которой расположен определенный объект.

Имена ссылочных типов данных начинаются с верхнего регистра.

Любой класс в Java - это ссылочный тип.

По умолчанию значение любой переменной ссылки - null.

<https://vk.com/dny_vin>

**19. 🔒 Массивы в Джава, объявление и инициализация массивов, длина массива, получение доступа к элементу массива**

<https://vk.com/individualo4nick>

Массив — это структура данных, в которой хранятся элементы одного типа. Его можно представить, как набор пронумерованных ячеек, в каждую из которых можно поместить какие-то данные (один элемент данных в одну ячейку). Доступ к конкретной ячейке осуществляется через её номер. Номер элемента в массиве также называют индексом.

**Объявление массива**

Как и любую переменную, массив в Java нужно объявить. Сделать это можно одним из двух способов. Они равноправны, но первый из них лучше соответствует стилю Java. Второй же — наследие языка Си (многие Си-программисты переходили на Java, и для их удобства был оставлен и альтернативный способ). В таблице приведены оба способа объявления массива в Java:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Объявление массива, Java-синтаксис | Примеры | Комментарий |
| 1. | dataType[] arrayName; | int[] myArray;  Object[]  arrayOfObjects; | Желательно объявлять массив именно таким способом, это Java-стиль |
| 2. | dataType arrayName[]; | int myArray[];  Object  arrayOfObjects[]; | Унаследованный от С/С++ способ объявления массивов, который работает и в Java |

Создание массива. Как и любой другой объект, создать массив Java, то есть зарезервировать под него место в памяти, можно с помощью оператора new. Делается это так:

new typeOfArray [length];

Где typeOfArray — это тип массива, а length — его длина (то есть, количество ячеек), выраженная в целых числах (int). Однако здесь мы только выделили память под массив, но не связали созданный массив ни с какой объявленной ранее переменной. Обычно массив сначала объявляют, а потом создают, например:

int[] myArray; // объявление массива

myArray = new int[10]; // создание, то есть, выделение памяти для массива на 10 элементов типа int

 Однако гораздо чаще массив создают сразу после объявления с помощью такого сокращённого синтаксиса:

int[] myArray = new int[10]; // объявление и выделение памяти “в одном флаконе”

Обратите внимание: После создания массива с помощью new, в его ячейках записаны значения по умолчанию. Для численных типов (как в нашем примере) это будет 0, для boolean — false, для ссылочных типов — null.  Таким образом мы получаем массив из десяти целых чисел, и, пока это не изменится в ходе программы, в каждой ячейке записан 0.

Длина массива в Java. Как мы уже говорили выше, длина массива — это количество элементов, под которое рассчитан массив. Длину массива нельзя изменить после его создания.

Получить доступ к длине массива можно с помощью переменной length. Пример:

int[] myArray = new int[10]; // создали массив целых чисел на 10 элементов

System.out.println(myArray.length); // вывели в консоль длину массива

Вывод программы: 10

Одномерные и многомерные Java массивы. Многомерный массив объявляется и создается следующим образом:

Int[][] myTwoDimentionalArray = new int [8][8];

Где два, там и три. В Java можно задать массив массивов… массив массивов массивов и так далее. Правда, трёхмерные и более массивы используются очень редко. Тем не менее, с помощью трёхмерного массива можно запрограммировать, например, кубик Рубика.

Полезные методы для работы с массивами. Для работы с массивами в Java есть класс java.util.Arrays

Сортировка массива

Метод void sort(int[] myArray, int fromIndex, int toIndex) сортирует массив целых чисел или его подмассив по возрастанию.

Поиск в массиве нужного элемента

int binarySearch(int[] myArray, int fromIndex, int toIndex, int key). Этот метод ищет элемент key в уже отсортированном массиве myArray или подмассиве, начиная с fromIndex и до toIndex. Если элемент найден, метод возвращает его индекс, если нет - (-fromIndex)-1.

Преобразование массива к строке

Метод String toString(int[] myArray) преобразовывает массив к строке. Дело в том, что в Java массивы не переопределяют toString(). Это значит, что если вы попытаетесь вывести целый массив (а не по элементам) на экран непосредственно (System.out.println(myArray)), вы получите имя класса и шестнадцатеричный хэш-код массива (это определено Object.toString()).

Пример на sort, binarySearch и toString

Давайте создадим массив целых чисел, выведем его на экран с помощью toString, отсортируем с помощью метода sort и найдём в нём какое-то число.

class Main {

    public static void main(String[] args) {

        int[] array = {1, 5, 4, 3, 7};

        System.out.println(array);

        System.out.println(Arrays.toString(array));

        Arrays.sort(array, 0, 4);

        System.out.println(Arrays.toString(array));

        int key = Arrays.binarySearch(array, 5);

        System.out.println(key);

System.out.println(Arrays.binarySearch(array, 0));

    }

}

Вывод программы:

[I@1540e19d

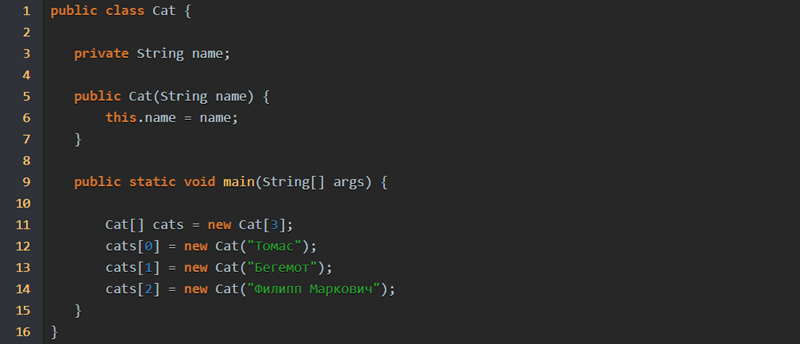
[1, 5, 4, 3, 7]

[1, 3, 4, 5, 7]

3

-1

**20. 🔒 Массивы в Джава, как объектные типы данных, контроль доступа за выход за границы массива**



Здесь нужно понимать несколько вещей:

В случае с примитивами массивы Java хранят множество конкретных значений (например, чисел int). В случае с объектами массив хранит множество ссылок. Массив cats состоит из трех ячеек, в каждой из которых есть ссылка на объект Cat. Каждая из ссылок указывает на адрес в памяти, где этот объект хранится.

Элементы массива в памяти размещаются в едином блоке. Это сделано для более эффективного и быстрого доступа к ним. Таким образом, ссылка cats указывает на блок в памяти, где хранятся все объекты — элементы массива. А cats[0] — на конкретный адрес внутри этого блока.

Контроль выхода за границы массива. Это делает интерпретатор, в случае выхода индекса за пределы массива будет инициировано исключение java.lang.ArrayIndexOutOfBoundsException. Перехватывать его не обязательно, и, я бы даже сказал, нежелательно, т.к. это RuntimeException и сигнализирует оно о том, что программа работает неправильно. И уж совсем не стоит проектировать приложение в расчете на то, что после прохождения всего массива будет брошено это исключение, и это будет сигналом к завершению цикла обработки. Это очень плохая идея.

**21. 🔒 Операции над массивами, просмотр элементов массива, поиск по образцу, сортировка массива, сумма элементов массива**

Получите первый и последний элемент массива

int firstItem = array[0];

int lastItem = array[array.length - 1];

Получить случайное значение из массива

Используя объект java.util.Random , мы можем легко получить любое значение из нашего массива:

int anyValue = array[new Random().nextInt(array.length)];

Добавить новый элемент в массив

Как мы знаем, массивы содержат фиксированный размер значений. Поэтому мы не можем просто добавить элемент и превысить это ограничение.

Нам нужно начать с объявления нового, большего массива и скопировать элементы базового массива во второй.

К счастью, класс Arrays предоставляет удобный метод для репликации значений массива в новую структуру разного размера:

int[]newArray = Arrays.copyOf(array, array.length + 1);

newArray[newArray.length - 1]= newItem;

Просмотр элементов массива

int[] intArray = { 1, 2, 3, 4, 5 };

String intArrayString = Arrays.toString(intArray);

System.out.println(intArray);  // [I@7150bd4d

System.out.println(intArrayString);  // [1, 2, 3, 4, 5]

Поиск по образцу

  int[] array = {1, 5, 4, 3, 7};

  int key = Arrays.binarySearch(array, 5);

  System.out.println(key);   // индекс элемента - 1

Или:

private boolean existA(String a) {

    for (String s : massStringA) {

        if (a.equals(s)) {

            return true;

        }

    }

    return false;

}

Сумма элементов массива

Используя Java 8 Stream API (сначала с помощью метода Arrays::stream преобразовываем массив в поток целых чисел, затем получаем сумму этого потока методом Stream::sum)

int array[] = {3, 5, 7, 12};

int sum = Arrays.stream(array).sum();

Или:

package ua.com.prologistic;

public class SumOfArray{

   public static void main(String args[]){

      int[] array = {10, 30, 20, 50, 40, 10};

      int sum = 0;

      for( int num : array) {

          sum = sum + num;

      }

      System.out.println("Сумма элементов массива равна: " + sum);

   }

}

Сортировка

import java.util.Arrays;

class ArraySort {

    public static void main(String[] args) {

        int[] arr = { 5, 2, 1, 8, 10 };

        Arrays.sort(arr);

        for (int values : arr) {

            System.out.print(values + ", ");    // 1, 2, 5, 8, 10,

        }

    }

}

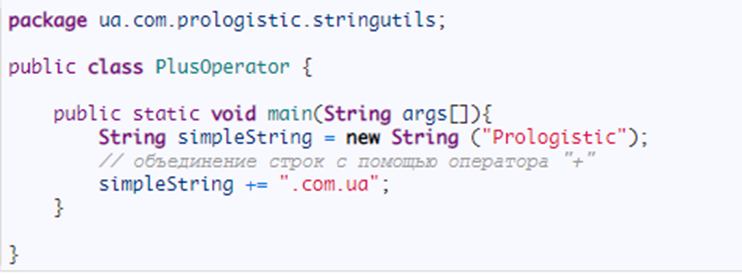
**22. 🔒 В результате выполнения фрагмента программы**

**23. 🔒 Операция конкатенации строк в Джава, ее обозначение и использование и ее использование**

Конкатенация строк - операция склеивания строк

При работе со строками, а именно конкатенацией, не желательно использовать оператор «+». Для этих целей следует использовать StringBuffer или StringBuilder.

*С помощью “+”:*



Конкатенация строк с помощью StringBuffer и StringBuilder будут аналогичны друг другу:



Когда мы объединяем строки с помощью оператора «+», происходит следующее:

1. Создается новый объект StringBuilder.

2. Строка «Prologistic» копируется в только что созданный объект StringBuilder.

3. Вызывается метод append() для добавления строки «.com.ua» к объекту StringBuilder.

4. Вызывается метод toString() для получения объекта типа String с объекта StringBuilder.

5. Ссылка на только что созданный объект типа String присваивается simpleString, а старая строка «Prologistic» становится доступной для сборщика мусора.

А что же происходит, если мы используем StringBuffer или StringBuilder:

1. Создается новый объект StringBuffer со значением «Prologistic».

2. Вызывается метод append() для добавления строки «.com.ua» к объекту.

3. Вызывается метод toString() для получения объекта типа String с объекта StringBuffer.

Судя по количеству действий, необходимых для конкатенации строк, способ с использованием StringBuffer или StringBuilder является менее трудоемким, использует меньше ресурсов и производит меньше мусора для уборщика мусора.

**24. 🔒 Циклы в Джава, цикл с предусловием, цикл с постусловием, пример записи и использование. Условие окончания цикла.**

Еще одним видом управляющих конструкций являются циклы. Циклы позволяют в зависимости от определенных условий выполнять определенное действие множество раз. В языке Java есть следующие виды циклов:

● for

● while

● do...while

Цикл с предусловием

Цикл while сразу проверяет истинность некоторого условия, и если условие истинно, то код цикла выполняется:

int j = 6;

while (j > 0){

    System.out.println(j);

    j--;

}

Так как перед выполнением тела цикла мы всегда предварительно вычисляем логическое выражение (условие входа в цикл), то этот вид while часто называют циклом с предусловием.

Цикл с постусловием

Цикл do сначала выполняет код цикла, а потом проверяет условие в инструкции while. И пока это условие истинно, цикл повторяется.

int j = 7;

do{

    System.out.println(j);

    j--;

}

while (j > 0);

**25. 🔒 Циклы в Джава, итерационный цикл for(), синтаксис, счетчик цикла, условие окончания цикла, модификация счетчика, пример использования,**

В языке Java есть следующие виды циклов:

● for

● while

● do...while

Цикл for имеет следующее формальное определение:

for ([инициализация счетчика]; [условие]; [изменение счетчика])

{

 // действия

 }

Пример:

for (int i = 1; i < 9; i++)

{ System.out.printf("Квадрат числа %d равен %d \n", i, i \* i);

}

Синтаксис цикла for

Возможные виды изменения счетчика: i++, i--, ++i, --i, i+/- const (где i – переменная-счетчик)

**26. 🔒 Способы объявления массивов в Джава, использование операции new для выделения памяти для элементов массива. Объявление с инициализацией, объявление массива определенного размера без инициализации.**

Способы объявления массивов в Джава

Как и любую переменную, массив в Java нужно объявить. Сделать это можно одним из двух способов. Они равноправны, но первый из них лучше соответствует стилю Java. Второй же — наследие языка Си (многие Си-программисты переходили на Java, и для их удобства был оставлен и альтернативный способ). В таблице приведены оба способа объявления массива в Java:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Объявление массива, Java-синтаксис | Примеры | Комментарий |
| 1. | dataType[] arrayName; | int[] myArray;  Object[]  arrayOfObjects; | Желательно объявлять массив именно таким способом, это Java-стиль |
| 2. | dataType arrayName[]; | int myArray[];  Object  arrayOfObjects[]; | Унаследованный от С/С++ способ объявления массивов, который работает и в Java |

В обоих случаях dataType — тип переменных в массиве. В примерах мы объявили два массива. В одном будут храниться целые числа типа int, в другом — объекты типа Object. Таким образом при объявлении массива у него появляется имя и тип (тип переменных массива). arrayName — это имя массива.

Как и любой другой объект, создать массив Java, то есть зарезервировать под него место в памяти, можно с помощью оператора new. Делается это так:

new typeOfArray [length];

Где typeOfArray — это тип массива, а length — его длина (то есть, количество ячеек), выраженная в целых числах (int). Однако здесь мы только выделили память под массив, но не связали созданный массив ни с какой объявленной ранее переменной. Обычно массив сначала объявляют, а потом создают, например:

int[] myArray; // объявление массива

myArray = new int[10]; // создание, то есть, выделение памяти для массива на 10 элементов типа int

Здесь мы объявили массив целых чисел по имени myArray, а затем сообщили, что он состоит из 10 ячеек (в каждой из которых будет храниться какое-то целое число). Однако гораздо чаще массив создают сразу после объявления с помощью такого сокращенного синтаксиса:

// ниже объявление массива определенного размера без инициализации.

int[] myArray = new int[10];

мы получаем массив из десяти целых чисел, и, пока это не изменится в ходе программы, в каждой ячейке записан 0. Для численных типов (как в нашем примере) это будет 0, для boolean — false, для ссылочных типов — null.

объявление массива определенного размера с инициализацией

int[] arr = {1, 2, 3};

Инициализация массива

Инициализация массива — это заполнение его конкретными данными (не по умолчанию).

int[] marks = new int[10];

marks[0] = 5;

marks[1] = 3;

**Тема 2. Реализация ООП в Java.**

**27. 🔒 Объявление класса на Джава, пример объявления**

<https://vk.com/id198165768>

Шаблоном или описанием объекта является класс, а объект представляет экземпляр этого класса. В данном случае класс называется Person. После названия класса идут фигурные скобки, между которыми помещается тело класса - то есть его поля и методы.

class Person{

    String name;        // имя

    int age;            // возраст

    void displayInfo(){

        System.out.printf("Name: %s \tAge: %d\n", name, age);

    }

}

**28.🔒Использования this для доступа к компонентам класса.** <https://vk.com/individualo4nick>

Ключевое слово this

1. Ссылка на текущий объект

Иногда требуется, чтобы метод ссылался на вызвавший его объект. Ключевое слово this в Java используется в теле любого метода для ссылки на текущий объект.

Рассмотрим конструктор, в котором параметры имеют те же имена, что и переменные класса. В этом случае параметры перекрывают область видимости переменных класса и мы не можем напрямую обратиться к переменным класса. Чтобы это сделать используется ключевое слово this:

Box(double width, double height, double depth) {

    this.width = width;

    this.height = height;

    this.depth = depth;

}

В этом примере использование ключевого слова this не является необходимым - можно обойтись и без него:

Box(double w, double h, double d) {

     this.width = w;

     this.height = h;

     this.depth = d;

}

2. Использование ключевого слова this() в конструкторе

Второй вариант использования ключевого слова this() - с его помощью можно вызвать один конструктор из другого. Вызов this() может находиться только в первой строчке конструктора:

public class Toy {

    String name;

    int cost;

    String manufacturer;

    int age;

    public Toy(String name, int cost, String manufacturer, int age) {

        this(name, cost, manufacturer);

        this.age = age;

        System.out.println("В конструкторе с четырьмя параметрами");

    }

    public Toy(String name, int cost, String manufacturer) {

        this();

        this.name = name;

        this.cost = cost;

        this.manufacturer = manufacturer;

        System.out.println("В конструкторе с тремя параметрами");

    }

    public Toy() {

        System.out.println("В конструкторе по умолчанию");

    }

}

public class ToyDemo {

    public static void main(String[] args) {

        Toy toy = new Toy("Кукла", 34, "Disney", 3);

    }

}

Результат программы:

В конструкторе по умолчанию

В конструкторе с тремя параметрами

В конструкторе с четырьмя параметрами

**29.🔒Создание или инстанцирование объектов типа класс:**

Инстанцирование (англ. instantiation) — создание экземпляра класса. В отличие от слова «создание», применяется не к объекту, а к классу. То есть, говорят: «(в виртуальной среде) создать экземпляр класса или инстанцировать класс». Порождающие шаблоны используют полиморфное инстанцирование.

Конструкция:

ClassName ObjectName = new classConstructor();

Пример:

Point originOne = new Point(23, 94);

Rectangle rectOne = new Rectangle(originOne, 100, 200);

Rectangle rectTwo = new Rectangle(50, 100);

Первая строка создает объект Point class, и вторые и третьи строки каждый создает объект Rectangle class.

У каждого из этих операторов есть три части:

Объявление: кодовый набор полужирным является всеми объявлениями переменной, которые связывают имя переменной с объектным типом.

Инстанцирование: ключевое слово new является оператором Java, который создает объект.

Инициализация: оператор new сопровождается звонком в конструктора, который инициализирует новый объект.

**30.🔒Что такое класс в Java?**

<https://vk.com/individualo4nick>

Класс в Java - это шаблон для создания объекта, а объект - это экземпляр класса. Класс определяет структуру и поведение, которые будут совместно использоваться набором объектов. Класс содержит переменные и методы, которые называются элементами класса, членами класса. Он составляет основу инкапсуляции в Java. Каждый объект данного класса содержит структуру и поведение, которые определены классом. Иногда объекты называют экземплярами класса.

Методы используются для описания того, что объект класса умеет делать или что можно с ним сделать. Переменные - для описания свойств или характеристик объекта.

Рассмотрим как создать класс в языке Java. Упрощенная общая форма определения класса:

class ИмяКласса{

       тип переменнаяЭкземпляра1;

       тип переменнаяЭкземпляра2;

         // ...

       тип переменнаяЭкземпляраN;

      тип имяМетода 1 ( список параметров) {

         // тело метода

       }

       тип имяМетода2 (список параметров) {

         // тело метода

        }

       …

      тип имяМетодаN (список  параметров ) {

        // тело метода

       }

}

После ключевого слова class пишется имя класса. В теле класса объявляются переменные и методы класса. Их может быть сколько угодно.

Опишем класс для объекта Box (коробка). У коробки есть три главные характеристики: ширина, высота и глубина, описанные с помощью переменных:

public class Box {

    double width;

    double height;

    double depth;

}

**31.🔒Модификатор доступа или видимости в Джава, виды и использование**

<https://vk.com/marenau>

В Джава существуют различные виды модификаторов видимости, их еще называют модификаторы доступа. Роль модификаторов доступа в программах на Джава обеспечить область видимости для данного компонента класса, в зависимости от того с каким модификатором он объявлен. Данные и методы класса, объявленные с public доступны в любом месте программы. Данные и методы класса, объявленные с private доступны внутри класса. Все компоненты класса, которые объявлены без модификатора видимости по умолчанию имеют видимость и могут быть доступны из любого класса в том же пакете.

Поля и методы, обозначенные модификатором доступа protected будут видны: в пределах всех классов, находящихся в том же пакете, что и наш; в пределах всех классов-наследников нашего класса.



**32.🔒Чем отличаются static-метод класса от обычного метода класса:**

Обычный метод имеет привязку к объекту — экземпляру класса, статический же метод такой привязки не имеет.

Обычный метод может обращаться к переменным в своем экземпляре класса, статический — нет: у него просто нет никакого экземпляра класса, связанного с ним.

https://vk.com/tenshi\_1

**33.🔒Для чего используется оператор new?**

Ключевое слово new отвечает за выделение памяти. Используется для создания объектов класса, а также массивов.

<https://vk.com/id253451617>

**34.🔒Можно ли вызвать static-метод внутри обычного метода?**

Можно, ничего дополнительно делать не надо.

<https://vk.com/id253451617>

**35.🔒Как вызвать обычный метод класса внутри static-метода?**

 сначала нужно создать объект этого класса, а только потом вызвать метод у объекта. Вызвать обычный метод не у объекта, а у класса нельзя.

https://vk.com/tenshi\_1

**36.🔒Для чего используется в Джава ключевое слово this?**

Как правило, применять ключевое слово this в Java надо в двух случаях:

1. Когда у переменной экземпляра класса и переменной метода/конструктора одинаковые имена;
2. Когда нужно вызвать конструктор одного типа (например, конструктор по умолчанию или параметризованный) из другого. Это еще называется явным вызовом конструктора.

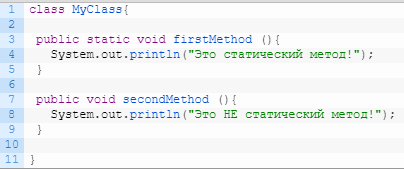
<https://vk.com/id253451617>

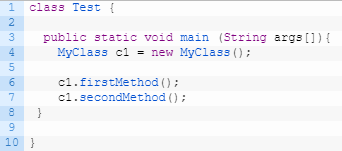
**37.🔒Объявление и использование методов, объявленных с модификатором public static**

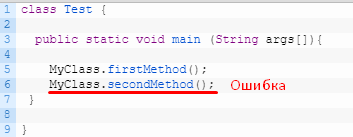
<https://vk.com/marenau>

Модификатор static (постоянный) делает переменную или метод независимым от объекта класса.

1. Методы с модификатором public static можно вызывать без создания объекта класса. К статическому и обычному методу можно обращаться через объекты класса, но через класс (без объекта) можно обращаться только к статическому методу.







1. Статические методы нельзя переопределить в классах наследниках. Такой метод един и для текущего класса и для всех его наследников.
2. Статические методы могут работать только со статическими методами или полями. Так как статические методы работают без создания объектов, а если в этих методах попытаться обратиться к нестатическому полю, то будет ошибка, так как объект не создавался, соответственно и поля тоже не существует. (Но через нестатические методы можно обращаться к статическим методам и полям).

**38.🔒Синтаксис объявления методов, тип возвращаемого значения, формальные параметры и аргументы**

<https://vk.com/marenau>

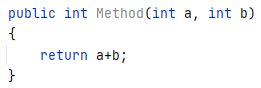
Синтаксис:

[модификаторы] <возвращаемый тип> <название> (<список формальных параметров>) [throws исключения] {<тело метода>}

или

[модификаторы] <возвращаемый тип> <сигнатура> [throws исключения] {<тело метода>}

(сигнатура - это имя и формальные параметры вместе).

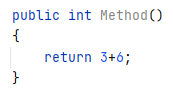


Формальные параметры - это переменные, перечисленные в описании метода - (a, b).

Аргументы - это фактические параметры (значения), передаваемые методу при его вызове - Method(5, 7).

**39.🔒Методы с пустым списком параметров**

<https://vk.com/marenau>

Методы без списка формальных параметров.

**40.🔒Стандартные методы класса сеттеры и геттеры, синтаксис и их назначение?**

<https://vk.com/marenau>

В Java геттер и сеттер - это два метода, которые используются для получения значения поля класса (геттер) или его изменения (сеттер).

Когда переменные имеют модификатор доступа private, то есть скрыты, то к ним нельзя обращаться извне текущего класса. В таком случае соблюдается инкапсуляция и посторонние не смогут произвольно менять поля объектов. Но для осуществления взаимодействия с закрытым полям есть геттеры и сеттеры.

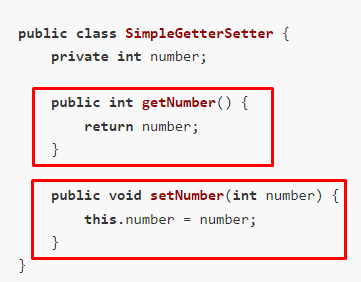
Геттер - это метод, который возвращает нам значение какого-то поля.

Сеттер - это метод, который изменяет значение поля (но изменяет по прописанным правилам, а не просто так).

Реализация геттеров и сеттеров обеспечивает инкапсуляцию в коде программы. Наличие геттеров нарушает неизменяемость.

Пару правил:

1. В геттерах не стоит возвращать ссылку на поле (например массив), так как с ссылкой можно менять закрытое поле. Для такого случая в геттере лучше создать новый пустой массив, все значения поля скопировать в созданный массив и вернуть ссылку именно на созданный массив, но с такими же значениями.
2. В сеттерах не стоит сразу полю объекта присваивать полученную ссылку. В таком случае тоже появляется возможность менять поля извне. Для такого случая лучше создать в сеттере пустой массив и присвоить его полю. А уже потом все значения полученного массива присвоить полю.



**41. 🔒 Может ли быть поле данных класса объявлено как с модификатором static и final одновременно и что это означает?**

<https://vk.com/l1senok>

Да, может. Тогда у поля будут следующие свойства:

1. static: поле относится к классу, а не к объекту
2. final: для примитивов: присвоенное значение не может быть изменено, для ссылочных переменных: нельзя изменить ссылку на данный объект, но его состояние изменить можно.

**42.🔒Методы класса, конструкторы, синтаксис и назначение**

Допустим, создадим класс кот, с некоторыми полями данных:

1. public class Cat {

2.     String color;

3.     int weight;

4.     String sex;

5.     void myaukat(){

6.   … }

7. void walk(){

8.   … }

9. }

Название объекта — название класса; свойства — поля класса и действия — это будут методы нашего класса.

В классе мы описали сам объект, его свойства и методы, но мы не создали конкретного кота. Для того, чтобы создавать конкретные объекты в джаве предусмотренное ключевое слово new и конструктор. По сути дела, конструктор — это тоже метод. Только он создан чтобы создавать объект. Правильная форма записи конструктора такая: ИмяКласса(){}. Это конструктор по умолчанию. Такой конструктор создаст нашего котенка, но животное будет без набора параметров. Для того, чтобы создать объект с параметрами можно применять конструктор с параметрами: ИмяКласса(параметр1, параметр2){ поле класса = параметр1; поле класса = параметр2;}. Конструкторов в классе можно создавать неограниченное количество и с разным набором параметров. Если Вы не создали конструктора, то будет использован конструктор по умолчанию: без параметров.

**43.🔒Может ли класс иметь в своем составе несколько конструкторов?**

<https://vk.com/cabbagetyan>

В одном классе может быть сразу несколько конструкторов. Создав несколько конструкторов мы можем передавать разное количество параметров при создании объекта. Компилятор сам определяет какой конструктор необходимо использовать в зависимости от передаваемых параметров и их типов данных.

Возможность указать в одном классе несколько конструкторов (или в общем смысле - методов с одинаковым названием) называется перегрузкой. Название у всех конструкторов в данном случае одинаковое, а вот типы и количество принимаемых параметров обязательно должны отличаться.

Пример класса с несколькими конструкторами представлен на Рисунке 1



Рисунок 1 - Класс с несколькими конструкторами

**44.🔒Может ли конструктор класса возвращать значение?**

Конструктор в Java не возвращает значения (как функция), но может принимать значения, необходимые для конструирования класса.

**Тема 3. Реализация наследования в программах на Джаве**

**45.🔒Наследование в Джава. Вид наследования и синтаксис Ключевое слово extends**

https://vk.com/cabbagetyan

С помощью наследования можно расширить функционал уже имеющихся классов за счет добавления нового функционала или изменения старого. Чтобы объявить один класс наследником от другого, надо использовать после имени класса-наследника ключевое слово extends, после которого идет имя базового класса.

Если в базовом классе определены конструкторы, то в конструкторе производного классы необходимо вызвать один из конструкторов базового класса с помощью ключевого слова super. Например, класс Person имеет конструктор, который принимает один параметр. Поэтому в классе Employee в конструкторе нужно вызвать конструктор класса Person. То есть вызов super(name) будет представлять вызов конструктора класса Person.

При вызове конструктора после слова super в скобках идет перечисление передаваемых аргументов. При этом вызов конструктора базового класса должен идти в самом начале в конструкторе производного класса. Таким образом, установка имени сотрудника делегируется конструктору базового класса.

Причем даже если производный класс никакой другой работы не производит в конструкторе, как в примере выше, все равно необходимо вызвать конструктор базового класса.

С ключевым словом implements связано чуть больше хитростей. Слово «имплементировать» можно понимать, как «реализовывать», а в тот самый момент, когда возникает слово «реализовывать», где-то недалеко появляются интерфейсы. Так вот конструкция public class Door implements Openable означает, что класс дверь реализует интерфейс «открывающийся». Следовательно класс должен переопределить все методы интерфейса. Главная фишка в том, что можно реализовывать сколь угодно много интерфейсов.

**46.🔒Что означает перегрузка метода в Java (overload)?**

В Java разрешается в одном и том же классе определять два или более метода с одним именем, если только объявления их параметров отличаются. Это называется перегрузкой методов.

Перегрузка методов является одним из способов поддержки полиморфизма в Java.

**47.🔒Что означает переопределение метода в Java (override)?**

<https://vk.com/cabbagetyan>

Производный класс может определять свои методы, а может переопределять методы, которые унаследованы от базового класса. (рис.1)

Перед переопределяемым методом указывается аннотация @Override. Данная аннотация в принципе не обязательна.

При переопределении метода он должен иметь уровень доступа не меньше, чем уровень доступа в базовом класса. Например, если в базовом классе метод имеет модификатор public, то и в производном классе метод должен иметь модификатор public.

Однако в данном случае мы видим, что часть метода display в Employee повторяет действия из метода display базового класса. Поэтому мы можем сократить класс Employee(рис.2)

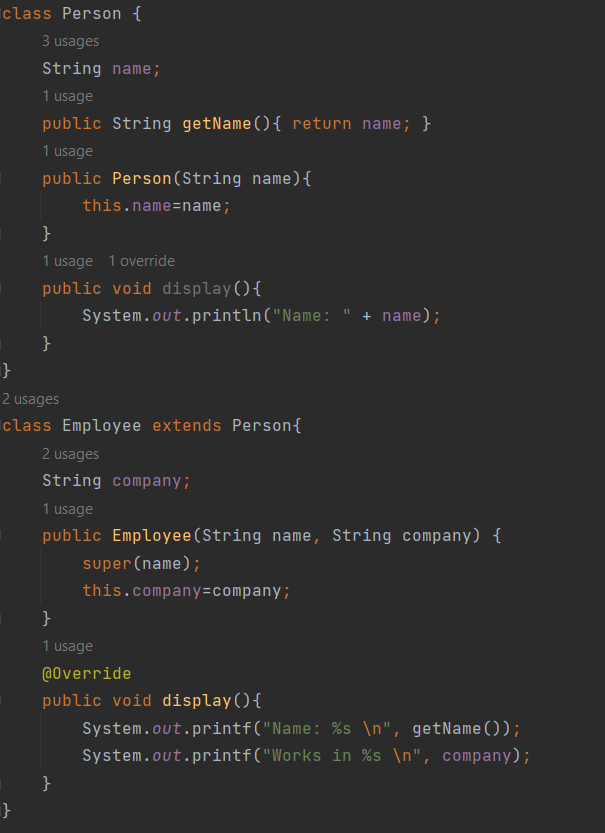


Рисунок 1 - Полное переопределение родительского метода

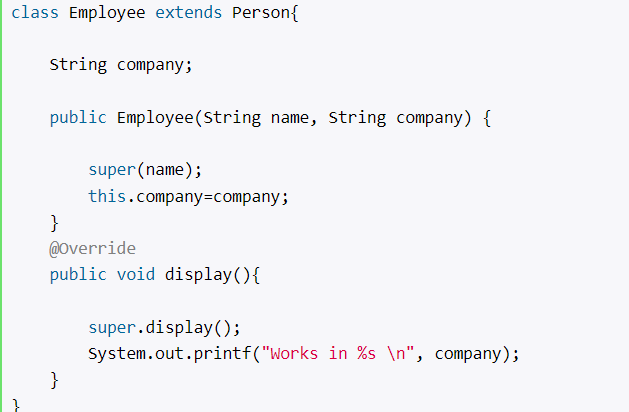


Рисунок 2 - Дополнение родительского метода

**48.🔒В чем разница между перегрузкой и переопределением методов, поясните**

Перегрузка метода имеет дело с понятием наличия двух или более методов в одном классе с одинаковым именем, но разными аргументами.

Переопределение метода означает наличие двух методов с одинаковыми аргументами, но разными реализациями. Один из них будет существовать в родительском классе, а другой-в производном или дочернем классе. Аннотация @Override , хотя и не является обязательной, может быть полезна для обеспечения надлежащего переопределения метода во время компиляции.

**49.🔒Абстрактные классы в Джава и абстрактные методы класса**

<https://vk.com/id229306625>

**Абстрактный класс** похож на обычный класс. В абстрактном классе также можно определить поля и методы, но в то же время нельзя создать объект или экземпляр абстрактного класса. Абстрактные классы призваны предоставлять базовый функционал для классов-наследников. А производные классы уже реализуют этот функционал.

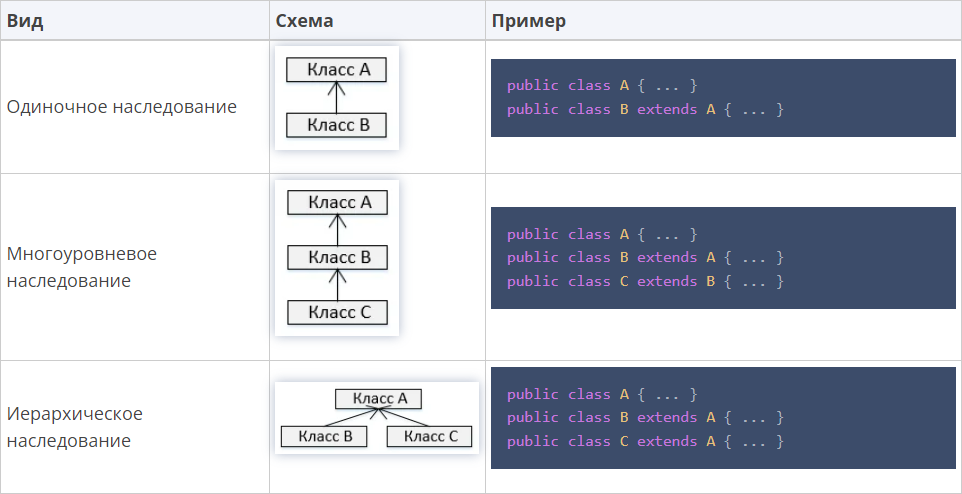
**Абстрактные методы** – это методы у которых отсутствует реализация.

Кроме обычных методов абстрактный класс может содержать **абстрактные методы**. Такие методы определяются с помощью ключевого слова abstract и не имеют никакой реализации.

Производный класс обязан переопределить и реализовать все абстрактные методы, которые имеются в базовом абстрактном классе. Также следует учитывать, что если класс имеет хотя бы один абстрактный метод, то данный класс должен быть определен как абстрактный.

**50.🔒Виды наследования в Джава, использование интерфейсов для реализации наследования**

<https://vk.com/id229306625>



Интерфейс Java представляет собой набор абстрактных методов и констант (необязательно).

Interface - служебное слово. Интерфейс не имеет определения (тела). Все методы интерфейса не имеют модификаторов доступа, но фактически по умолчанию доступ public. Класс формально реализует интерфейс:

* Заявив об этом, поэтому в заголовке класса написано implements.
* И обеспечивая реализацию каждого абстрактного метода интерфейса.

Если класс утверждает, что он реализует интерфейс, то он должен определить все методы в интерфейсе (создать их реализацию – тело).

 Интерфейсы могут наследоваться от любого множества интерфейсов, а не от одного как классы.

**51.🔒Что наследуется при реализации наследования в Джава (какие компоненты класса), а что нет?**

<https://vk.com/id229306625>

Java не поддерживает наследование нескольких классов. Один класс - один родитель.

Наследуется все кроме приватных переменных и методов. все методы и переменные, помеченные модификатором **private**, недоступны классу-наследнику.

**52.🔒К каким методам и полям базового класса производный класс имеет доступ (даже если базовый класс находится в другом пакете), а каким нет? Область видимости полей и данных из производного класса**

<https://vk.com/id229306625>

Производный класс имеет доступ ко **всем** методам и полям базового класса (даже если базовый класс находится в другом пакете) кроме тех, которые определены с модификатором private.



**Тема 4. Полиморфизм в Джава. Работа со строками. Интерфейсы.**

**53. 🔒 Объявление и инициализация переменных типа String**

<https://vk.com/artemvlastyuk>

Пример:

String a = “Hello”;

String b = “Hello world”.

При объявлении переменной, в следующей последовательности указываются:

– тип данных (в данном примере — String — переменная содержит строку),

– имя переменной (в данном примере имена — a и b),

– начальное значение переменной или, другими словами, инициализация переменной.

В данном примере переменным a и b присвоены строки “Hello” и “Hello world”. Однако, это не является обязательным условием при объявлении переменной.

Пример: объявление переменных без инициализации:

String a;

String b.

После каждой строки при объявлении переменных необходимо ставить точку с запятой “;”.

Пример: инициализация ранее объявленной переменной:

a = “Hello”;

b = “Hello world”.

**54. 🔒 Операция конкатенации строк и ее использование**

<https://vk.com/artemvlastyuk>

Конкатенация строк (сцепление) — это операция объединения строк. Для этой операции в Java используется знак +. К строке можно присоединять не только другую строку, но и значение любого другого типа, которое будет преобразовано к строке автоматически. Как же понять в каких случаях знак + это конкатенация, а в каких случаях арифметическая операция?

Важно запомнить всего два правила:

·   Если один из операндов типа String, то оператор + используется как конкатенация строк.

· Если оба операнда являются числами, то оператор + используется в качестве сложения.

В этом примере просто объединяем две строки:

public class StringExample1 {

    public static void main(String[] args) {

        String str1 = "world";

        System.out.println("Hello " + str1);

        System.out.println("First line\n" + "Second line");

    }

}

В следующем примере в первом System.out.println на консоль выведется строка "X Y". Мы объединяем char со String - это конкатенация. Во втором же System.out.println на консоль выведется не строка, как многие ожидают, а число 177. Почему число? Тип char является псевдоцелочисленным типом, и может участвовать в арифметических операциях. В данном случае складываются кодысимволов:

public class StringExample2 {

    public static void main(String[] args) {

    char x, y;

     x = 88;  // Код символа 'X'

     y = 'Y';

     System.out.println(x + " " + y); //Выведется X Y

     System.out.println(x + y); //Выведется 177

    }

}

**55. 🔒 Что означает утверждение, что объект класса String является неизменяемым** <https://vk.com/artemvlastyuk>

Объекты String являются неизменяемыми, поэтому все операции, которые изменяют строки, фактически приводят к созданию новой строки, что сказывается на производительности приложения.

В Java есть объекты, и есть ссылки на объекты. Когда вы пишете

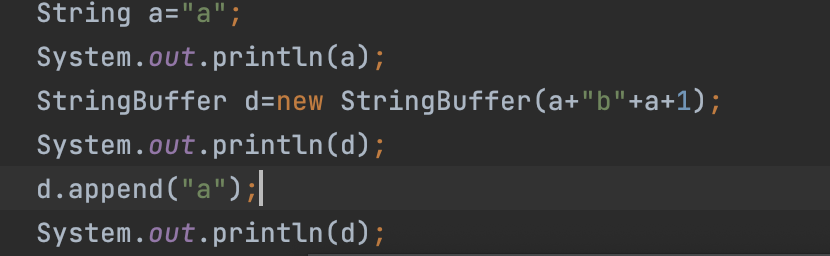
String s = "hello funny world";

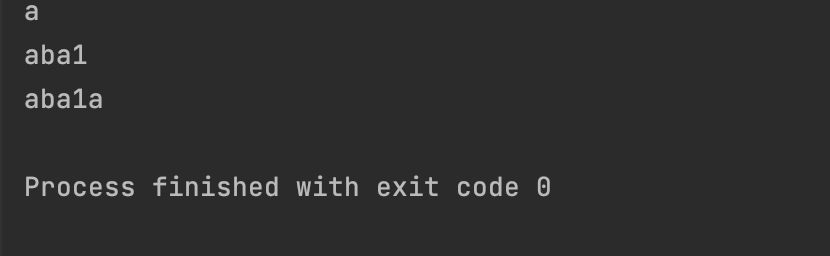
ваше s -- ссылка на объект-строку. Сам объект никогда не меняется. Чтобы "изменить" строку, вам придётся создать *новую* строку, и присвоить s ссылку на эту новую строку.

**56. 🔒 При создании объектов строк с помощью класса StringBuffer, например StringBuffer strBuffer = new StringBuffer(str) можно ли использовать операцию конкатенации строк или необходимо использовать методы класса StringBuffer**

<https://vk.com/artemvlastyuk>

Можно при создании. Потом чтобы конкатенировать строки надо использовать метод append



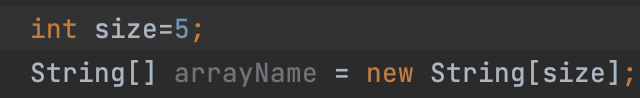


Конкатенация строк через StringBuffer в Java выполняется с помощью метода append.

**57. 🔒 Объявление и инициализация массива строк. Организация просмотра элементов массива**

<https://vk.com/artemvlastyuk>

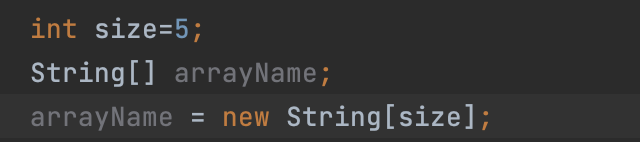
Общая форма объявления и выделение памяти для одномерного массива строк



где

* **String** – встроенный в Java класс, который реализует строку символов.
* ***arrayName*** – имя объекта (экземпляра) типа String. Фактически, *arrayName* есть ссылкой на объект типа String;
* ***size*** – размер массива (количество строк, количество элементов типа String).

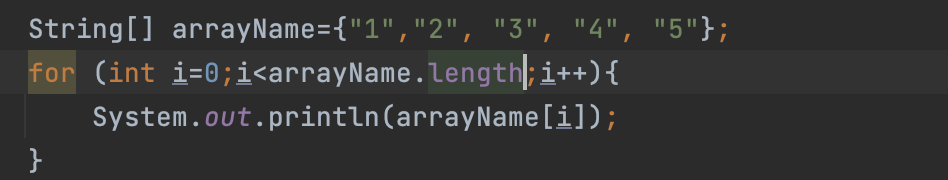
Объявление одномерного массива строк и выделение памяти для него можно реализовать и по другому:



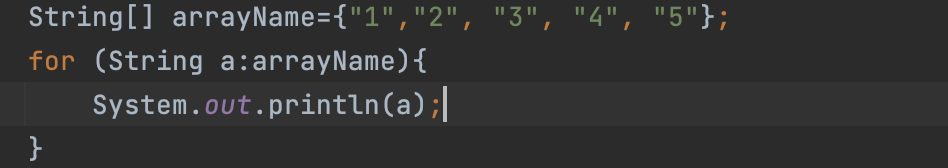
Ниже приведен пример инициализации одномерного массива строк.

https://lh4.googleusercontent.com/HoUjbJZf2t-q_q6wKqIyvXjMsCARaMRiDgOYlFU4b6JrOe9ZBpPl5vGqa5tzouSLha4w1fKlhFTDKESWgXo0jCVJvtQFaiB1kduIlfj3dJ8Q-AyuaWo8L_nSxnsnxMXAPpJTHEJEoZDuD8sZW65NMgpWZ-SIso-IKNq7sm6gCICWRRjG9NBrGTQqeVEvxAhttps://lh5.googleusercontent.com/-XGdl_qrIySu1OQvxRXtrp0DGSjxIfNQlfmE35tWxbgHnEUGAi0oaArUPlv_uHDlgUsJYnr8tRJAqi9ia7rxQyRH74RpReUqO_DdeRuIsG6Nl0pDGY7O67w2nx-lxl42si5pc8_a5WIiXg7eq3Q0VWTJFntFAEUdxEOfB-jZN-zcZjbtRxApgYDJo_4CdA

Просмотр элементов массива:



Другой способ:

****

**https://lh6.googleusercontent.com/NwH3L_bi8x4KMDv3wn4QuX_AzFcV5dTHdc5_zA3cdZt-f6xnp8PXn_0mQCqeYl3UeaHSuyDWDYnoUWxHL4lz3_FfxB3Or19IezVHsX2abVo5aGC_izfX3kA7uUnyuPCQ0ZQrsQ9qvefwy6axqfbP-aPxTSCXdqGWKrIqHWBO8SS6qSHefrjIvqfpKAskpg**

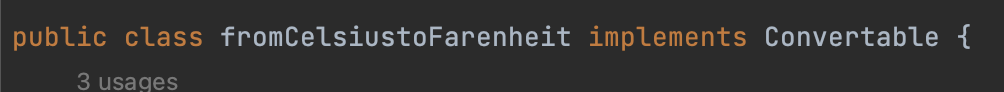
**58.🔒Понятие и объявление интерфейсов в Джава**

**Интерфейс в языке программирования Java** - это абстрактный тип, который используется для указания поведения, которое должны реализовывать классы.

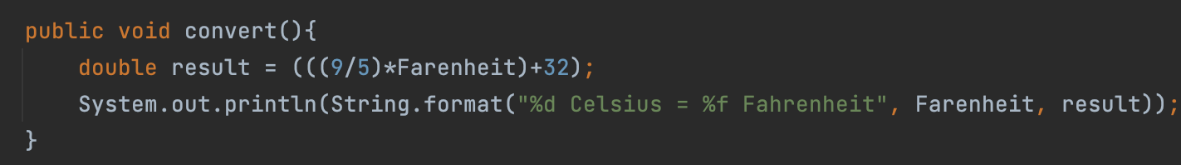
Создание интерфейса очень похоже на создание класса:



Чтобы подключить интерфейс к классу требуется приписывать **implements название**

****

Далее у нас появляется возможность реализовать **все**(это обязательное условие) методы интерфейса



<https://vk.com/dny_vin>

**59.🔒Может ли один класс реализовывать несколько интерфейсов?**

<https://vk.com/marenau>

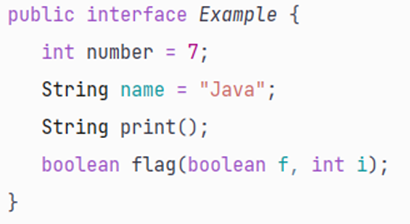
Да, класс может реализовать несколько интерфейсов. Интерфейсы перечисляются в списке, который следует за implements через запятую. Класс должен реализовывать все методы всех интерфейсов, перечисленных в заголовке.

https://lh4.googleusercontent.com/Cf69rNt19H2mET-889-hftNf7_Og1YBHWFcn8c0K6MMiwDrMSqZ7NCqQ-QL0uVZZorHP8FL3EnQFb4nl0pKy8Qr6Jp_wcgnEMl4U2oTqdkHBsH0gnUjnSXmzYzoAlWiztUf_bhw3jgOG9hK0RviczFN-hqK2i0kWHHx-QBeecl6WMRbpA5tcwSVEgkOPKw

**60.🔒Что входит в состав интерфейса. (какие компоненты может содержать интерфейс)?**

<https://vk.com/marenau>

В состав интерфейса входят абстрактные методы (методы, не имеющие реализации), а также константы (необязательно).



Interface - служебное слово. Ни один из методов интерфейса не имеет определения (тела). Методы в интерфейсе имеют модификатор public по умолчанию.

**61.🔒Может ли интерфейс наследоваться от другого интерфейса?**

Да.

Интерфейс может расширяться от одного или нескольких интерфейсов. Класс может расширять только один класс, но реализовывать любое количество интерфейсов

Интерфейс не может реализовать другой интерфейс. Он должен расширить другой интерфейс, если это необходимо.

**62.🔒Интерфейс Comparable, назначение, его методы и использование в Джава**

В интерфейсе Comparable объявлен только один метод compareTo (Object obj), предназначенный для упорядочивания объектов класса. Данный метод удобно использовать для сортировки списков или массивов объектов.

Метод compareTo (Object obj) сравнивает вызываемый объект с obj. В отличие от метода equals, который возвращает true или false, compareTo возвращает:

● 0, если значения равны;

● Отрицательное значение (обычно -1), если вызываемый объект меньше obj;

● Положительное значение (обычно +1), если вызываемый объект больше obj.

Если типы объектов не совместимы при сравнении, то compareTo (Object obj) может вызвать исключение ClassCastException. Необходимо помнить, что аргумент метода compareTo имеет тип сравниваемого объекта класса.

Обычные классы Byte, Short, Integer, Long, Double, Float, Character, String уже реализуют интерфейс Comparable.

Пример реализации интерфейса Comparable

package test;

import java.util.TreeSet;

class Compare implements Comparable<Object>

{

    String  str;

    int     num;

    String  TEMPLATE = "num = %d, str = '%s'";

    Compare(String str, int num)

    {

        this.str = str;

        this.num = num;

    }

    @Override

    public int compareTo(Object obj)

    {

        Compare entry = (Compare) obj;

        int result = str.compareTo(entry.str);

        if(result != 0)

            return result;

        result = num - entry.num;

        if(result != 0)

            return (int) result / Math.abs( result );

        return 0;

    }

    @Override

    public String toString()

    {

        return String.format(TEMPLATE, num, str);

    }

}

public class Example

{

    public static void main(String[] args)

    {

        TreeSet<Compare> data = new TreeSet<Compare>();

        data.add(new Compare("Начальная школа"  , 234));

        data.add(new Compare("Начальная школа"  , 132));

        data.add(new Compare("Средняя школа"    , 357));

        data.add(new Compare("Высшая школа"     , 246));

        data.add(new Compare("Музыкальная школа", 789));

        for (Compare e : data)

            System.out.println(e.toString());

    }

}

Результат выполнения программы:

num = 246, str = 'Высшая школа'

num = 789, str = 'Музыкальная школа'

num = 132, str = 'Начальная школа'

num = 234, str = 'Начальная школа'

num = 357, str = 'Средняя школа'

В примере значения сортируются сначала по полю str (по алфавиту), а затем по num в методе compareTo. Это хорошо видно по двум строкам с одинаковыми значения str и различными num. Чтобы изменить порядок сортировки значения str (в обратном порядке), необходимо внести небольшие изменения в метод compareTo.

@Override

public int compareTo(Object obj)

{

    int result = ((Comp)obj).str.compareTo(str);

    if(result != 0)

        return result;

    result = entry.number - number;

    if(result != 0) {

        return (int) result / Math.abs(result);

    return 0;

}

**63.🔒Какое значение возвращает вызов метода object1.compareTo(object2), который сравнивает 2 объекта obj1 и obj2 в зависимости от объектов?**

Метод compareTo (Object obj) сравнивает вызываемый объект с obj. В отличие от метода equals, который возвращает true или false, compareTo возвращает:

● 0, если значения равны;

● Отрицательное значение (обычно -1), если вызываемый объект меньше obj;

● Положительное значение (обычно +1), если вызываемый объект больше obj.

**64.🔒Интерфейсные ссылки и их использование в Джава**

Ранее неоднократно отмечалось, что в Java запрещено множественное наследование. Причина отказа от множественного наследования связана с теми потенциальными проблемами, которые могут при этом возникать. Однако множественное наследование открывает широкие перспективы для составления эффективных программных кодов и значительно повышает гибкость программ. Выход был найден в использовании интерфейсов.

Интерфейсы во многом напоминают классы. Принципиально от класса интерфейс отличается тем, что содержит только сигнатуры методов без описания, а также поля-константы (поля, значения которых постоянны и не могут изменяться).

Описание интерфейса аналогично к описанию класса, только ключевое слово class необходимо заменить ключевым словом interface. Как отмечалось, для методов интерфейса указываются только сигнатуры. Описываемые в интерфейсе поля по умолчанию считаются неизменяемыми (как если бы они были описаны с ключевым словом final) и статическими (то есть static). Таким образом, поля интерфейса играют роль глобальных констант. Практическое использование интерфейса подразумевает его реализацию. Эта процедура напоминает наследование абстрактных классов. Реализуется интерфейс в классе. Класс, который реализует интерфейс, должен содержать описание всех методов интерфейса. Методы интерфейса при реализации описываются как открытые. Один и тот же класс может реализовать одновременно несколько интерфейсов, равно как один и тот же интерфейс может реализовываться несколькими классами.

При создании объектов класса в качестве типа объектной переменной может указываться имя реализованного в классе интерфейса. Другими словами, если класс реализует интерфейс, то ссылку на объект этого класса можно присвоить интерфейсной переменной — переменной, в качестве типа которой указано имя соответствующего интерфейса. Через интерфейсную ссылку можно сослаться не на все члены объекта реализующего интерфейс класса. Доступны только те методы, которые объявлены в соответствующем интерфейсе. С учетом того, что класс может реализовать несколько интерфейсов, а один и тот же интерфейс может быть реализован в разных классах, ситуация представляется достаточно пикантной.

**Тема 5. Основные принципы и типы исключительных ситуаций.**

**65.🔒Понятие исключительной ситуации и ее обработка**

<https://vk.com/leo248>

**Исключительная ситуация** – это событие, которое привело к сбою в работе программы. В результате возникновения исключительной ситуации программа не может корректно продолжить свое выполнение.

**Обработка** исключительных ситуаций самой программой заключается в том, что при возникновении исключительной ситуации управление передается некоторому заранее определенному обработчику — блоку кода, процедуре, функции, которые выполняют необходимые действия.

**66.🔒В каком случае программа должна использовать оператор throw?**

<https://vk.com/leo248>

Оператор throws используется в объявлении метода для того, чтобы сообщить программе о том, что данный метод может генерировать исключение, которое он не обрабатывает.

Это касается всех исключений кроме: классов Error и RuntimeException; любых подклассов, которые унаследованы от классов Error и RuntimeException.

Основная информация Инструкция throw позволяет генерировать исключения, определяемые пользователем. При этом выполнение текущей функции будет остановлено (инструкции после throw не будут выполнены), и управление будет передано в первый блок catch в стеке вызовов.

**67.🔒В Java все исключения делятся на два основных типа. Что это за типы и какие виды ошибок ни обрабатывают?**

<https://vk.com/leo248>

Все исключения делятся на 2 вида — **проверяемые (checked)** и **непроверяемые (unchecked)**.

Все **проверяемые исключения (checked)** происходят от класса Exception. компилятор Java знает о них как о самых распространенных исключениях, и знает, в каких ситуациях они могут возникнуть.

Например, он знает, что если программист в коде считывает данные из файла, может легко возникнуть ситуация, что файл не существует. И таких ситуаций, которые он может заранее предположить, очень много.

Поэтому компилятор заранее проверяет наш код на наличие потенциальных исключений. Если он их найдет, то не скомпилирует код, пока мы не обработаем их или не пробросим наверх.

Второй вид исключений — “**непроверяемые**”.

Они происходят от класса RuntimeException. Этих ошибок компилятор не ожидает.

Он как бы говорит: “На момент написания кода я ничего подозрительного не обнаружил, но при его работе что-то пошло не так. Видимо, в коде есть ошибки!”

И это действительно так. Непроверяемые исключения чаще всего являются следствием ошибок программиста.

Примеры:

* ArithmeticException возникает при делении на ноль
* ArrayIndexOutOfBoundsException возникает при попытке обратиться к ячейке за пределами массива.

**68.🔒Код ниже вызовет ошибку: Exception <...> java. lang.ArrayIndexOutOfBoundsException: 4: Что она означает?**

<https://vk.com/leo248>

ArrayIndexOutOfBoundsException – это исключение, появляющееся во время выполнения. Оно возникает тогда, когда мы пытаемся обратиться к элементу массива по отрицательному или превышающему размер массива индексу.

**69.🔒Контролируемые исключения (checked)**

<https://vk.com/leo248>

Все **проверяемые исключения (checked)** происходят от класса Exception. компилятор Java знает о них как о самых распространенных исключениях, и знает, в каких ситуациях они могут возникнуть.

Например, он знает, что если программист в коде считывает данные из файла, может легко возникнуть ситуация, что файл не существует. И таких ситуаций, которые он может заранее предположить, очень много.

Поэтому компилятор заранее проверяет наш код на наличие потенциальных исключений. Если он их найдет, то не скомпилирует код, пока мы не обработаем их или не пробросим наверх.

**70.🔒Неконтролируемые исключения (unchecked) и ошибки, которые они обрабатывают**

<https://vk.com/leo248>

Второй вид исключений — “**непроверяемые**”.

Они происходят от класса RuntimeException. Этих ошибок компилятор не ожидает.

Он как бы говорит: “На момент написания кода я ничего подозрительного не обнаружил, но при его работе что-то пошло не так. Видимо, в коде есть ошибки!”

И это действительно так. Непроверяемые исключения чаще всего являются следствием ошибок программиста.

Примеры:

* ArithmeticException возникает при делении на ноль
* ArrayIndexOutOfBoundsException возникает при попытке обратиться к ячейке за пределами массива.

**71. 🔒Как реализуется принципы ООП в Java при создании исключений?**

Java является объектно-ориентированным языком. Это означает, что писать программы на Java нужно с применением объектно-ориентированного стиля. И стиль этот основан на использовании в программе объектов и классов.

Основные принципы ООП:

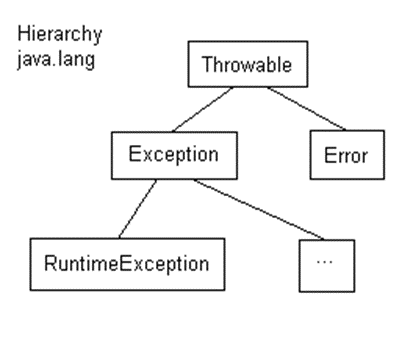
● Инкапсуляция

● Наследование

● Полиморфизм

В языке Java исключения (Exceptions) и ошибки (Errors) являются объектами. Когда метод вызывает, еще говорят "бросает" от слова "throws", исключительную ситуацию, он на самом деле работает с объектом. Но такое происходит не с любыми объектами, а только с теми, которые наследуются от Throwable.

Упрощенную диаграмму классов ошибок и исключений вы можете увидеть на следующем рисунке:



RuntimeException, Error и их наследников еще называют unchecked exception, а всех остальных наследников класса Exception - checked exception.

**72.🔒Какой оператор позволяет принудительно выбросить исключение?**

<https://vk.com/leo248>

Оператор throws используется в объявлении метода для того, чтобы сообщить программе о том, что данный метод может генерировать исключение, которое он не обрабатывает.

Это касается всех исключений кроме: классов Error и RuntimeException; любых подклассов, которые унаследованы от классов Error и RuntimeException.

Основная информация Инструкция throw позволяет генерировать исключения, определяемые пользователем. При этом выполнение текущей функции будет остановлено (инструкции после throw не будут выполнены), и управление будет передано в первый блок catch в стеке вызовов.

**73.🔒 Порядок выполнения операторов при обработке блока блока try...catch**

<https://vk.com/nadidal>

Блок кода, который мы ходим проверить необходимо обернуть в блок try. Далее, если выбросится то или иное исключение, оно будет передано в блок catch в зависимости от того, какое исключения было выброшено. Если в try поймалось исключение, код находящийся ниже исключения выполнен не будет никогда. Есть также блок finally, который выполняется вне зависимости от try и catch и стоит после них.

try { url = new URL(siteUrl); }

        catch (MalformedURLException e) {

            throw new RuntimeException(e);

        }

**Тема 6. Дженерики и использование контейнерных классов в Джава**

**74.🔒Абстрактный тип данных Stack (cтек) в Java**

<https://vk.com/id229306625>

Абстрактные типы данных- контейнеры

 Стеком (в переводе с английского – stack – стопка) называется линейная динамическая структура данных, добавление и исключение элементов в которую и производится с одного

конца, называемого вершиной стека.

Стек работает по принципу LIFO (Last-In, First-Out) - "поступивший последним, обслуживается первым».

Примеры применения стека — любая рекурсивная задача (“так, старую итерацию пока отложу в стопку, а сейчас надо обрабатывать новую итерацию!“),

Основными операциями над стеками являются:

* добавление элемента - push()
* удаление элемента - pop()

Абстрактные типы данных (АТД) или структуры данных

Абстрактный тип данных (ADT) является абстракцией структуры данных. АТД определяет:

* Хранение данных
* Операции выполняемые над данными
* Условия возникновения ошибок, связанных с выполнением операций

**75.🔒Универсальные типы или обобщенные типы данных, для чего создаются?**

<https://vk.com/id229306625>

В JDK введены так называемые обобщенные или параметризованные типы – generics или по-другому обобщенные типы.

Использование параметризации позволяет создавать классы, интерфейсы и методы, в которых тип обрабатываемых данных задается как параметр.

Дженерики или обобщенные типы позволяют вам абстрагироваться от использования конкретных типов.

Конвенция кода Java об именах для формальных типов

Например:

* <E> для элемента коллекции;
* <T> для обобщенного типа;
* <K, V> ключ и значение.
* <N> для чисел
* S,U,V, и т.д. для второго, третьего, четвертого типа параметра

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Обобщённые методы или дженерики позволяют работать с различными типами данных без изменения их описания, одно из назначений этих методов — более сильная проверка типов во время компиляции и устранение необходимости явного приведения. Создаётся более безопасный и легко читаемый код, который не перегружен переменными типа Object и приведением классов.

**76.Объявление обобщенного класса коллекции с параметризованным методом для обработки массива элементов коллекции на основе цикла for each (определение общего метода для отображения элементов массива)**

**77.🔒 Что представляет из себя класс ArrayList и в каком случае используется**

 (Мазурова)

ArrayList — реализация изменяемого массива интерфейса List, часть Collection Framework, который отвечает за список (или динамический массив), расположенный в пакете java.utils. В основе ArrayList лежит идея динамического массива. А именно, возможность добавлять и удалять элементы, при этом будет увеличиваться или уменьшаться по мере необходимости.

Что хранит ArrayList?

Только ссылочные типы, любые объекты, включая сторонние классы. Строки, потоки вывода, другие коллекции. Для хранения примитивных типов данных используются классы-обертки.

ArrayList следует использовать, когда в приоритете доступ по индексу, так как эти операции выполняются за константное время. Добавление в конец списка в среднем тоже выполняется за константное время. Кроме того в ArrayList нет дополнительных расходов на хранение связки между элементами. Минусы в скорости вставки/удаления элементов находящихся не в конце списка, так как при этой операции все элементы правее добавляемого/удаляемого сдвигаются.

**78. 🔒 Класс Pattern и его использование**

(мазурова)

Пакет java.util.regex предоставляется Java с целью сопоставления регулярных выражений с шаблоном. Пакет java.util.regex исходно состоит из следующих трех классов:

* Pattern Class – объект класса Pattern представляет скомпилированное представление регулярного выражения. В классе Pattern публичный конструктор не предусмотрен. Для создания шаблона, вам сперва необходимо вызвать один из представленных публичных статичных методов compile(), который далее произведет возврат объекта класса Pattern. Регулярное выражение в данных методах принимается как первый аргумент.
* Matcher Class
* PatternSyntaxException

Пример:



**79. 🔒 Класс Math и его использование**

 (мазурова)

Класс Math располагается в пакете java.lang и предоставляет набор статических методов для осуществления ряда различных математических вычислений.

Ниже приведены примеры вычислений, для которых класс Math может оказаться полезным:

* Вычисление абсолютных значений Math.abs(-1)
* Вычисление значений тригонометрических функций Math.sin(Math.toRadians(30))
* Возведение в различные степени Math.pow(3,2)//9
* Извлечение корней различных степеней Math.cbrt(125) //5
* Генерация случайных чисел Math.random() - от 0.0 до 1.0
* Округления

Метод round - округляет так, как привычно обывателю. Если дробная часть числа больше либо равна 0.5, то число будет округлено в большую сторону, иначе в меньшую.

Метод floor всегда, независимо от значений дробной части, округляет число в меньшую сторону (в сторону отрицательной бесконечности).

Метод ceil, напротив, вне зависимости от значений дробной части, округляет числа в большую сторону (в сторону положительной бесконечности).

**80. 🔒 Как вызываются методы класса Math и что при этом происходит?**

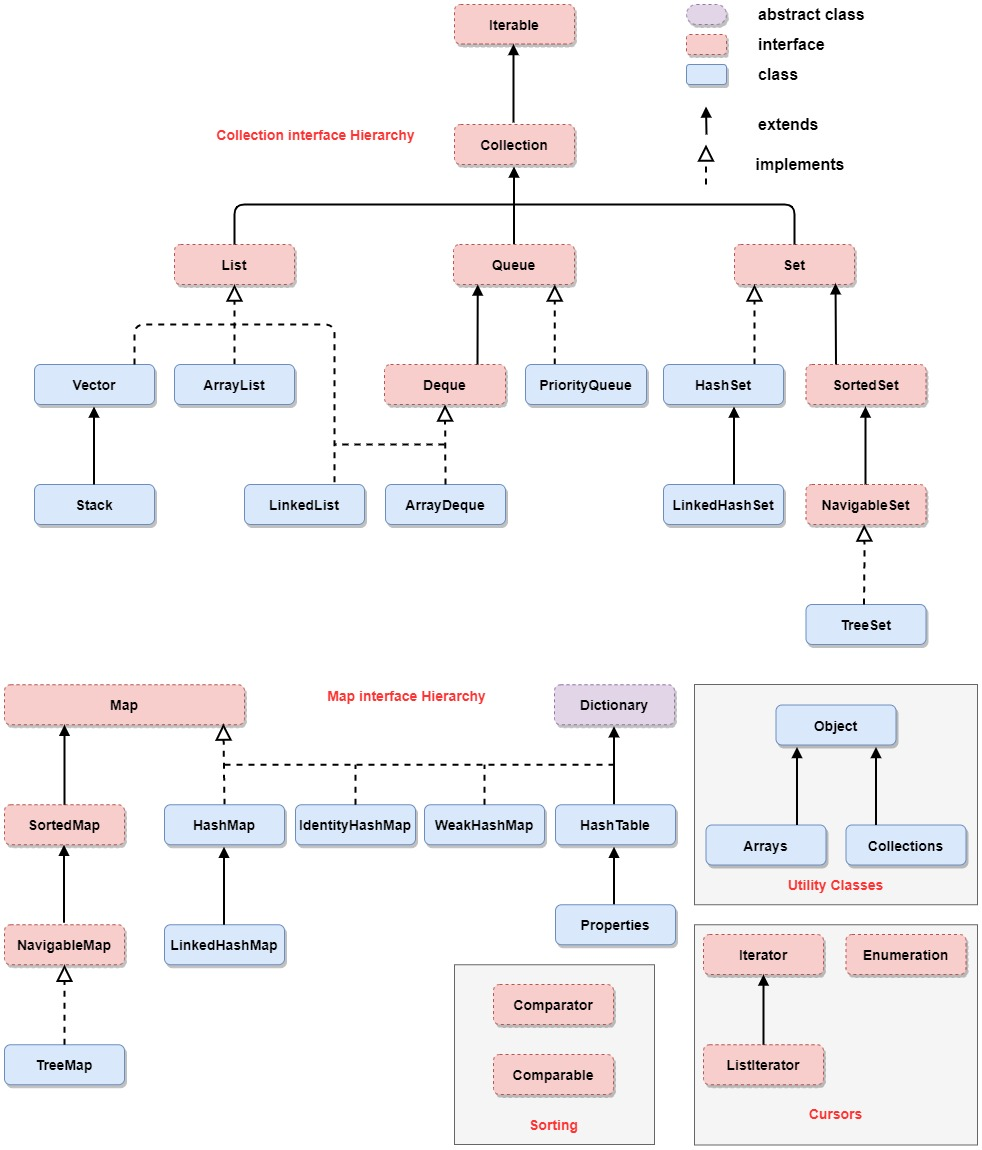
(мазурова)

Класс Math содержит методы, связанные с геометрией и тригонометрией и прочей математики. Методы реализованы как static, поэтому можно сразу вызывать через Math.methodName() без создания экземпляра класса.

**Тема 7. Java Core. Дженерики (продолжение) и использование контейнерных классов Java Framework Collection**

**81.🔒Структура коллекций в Java Collection Framework. Иерархия интефейсов**

<https://vk.com/marenau>



**82.🔒Коллекция HashMap, создание и методы работы с ней**

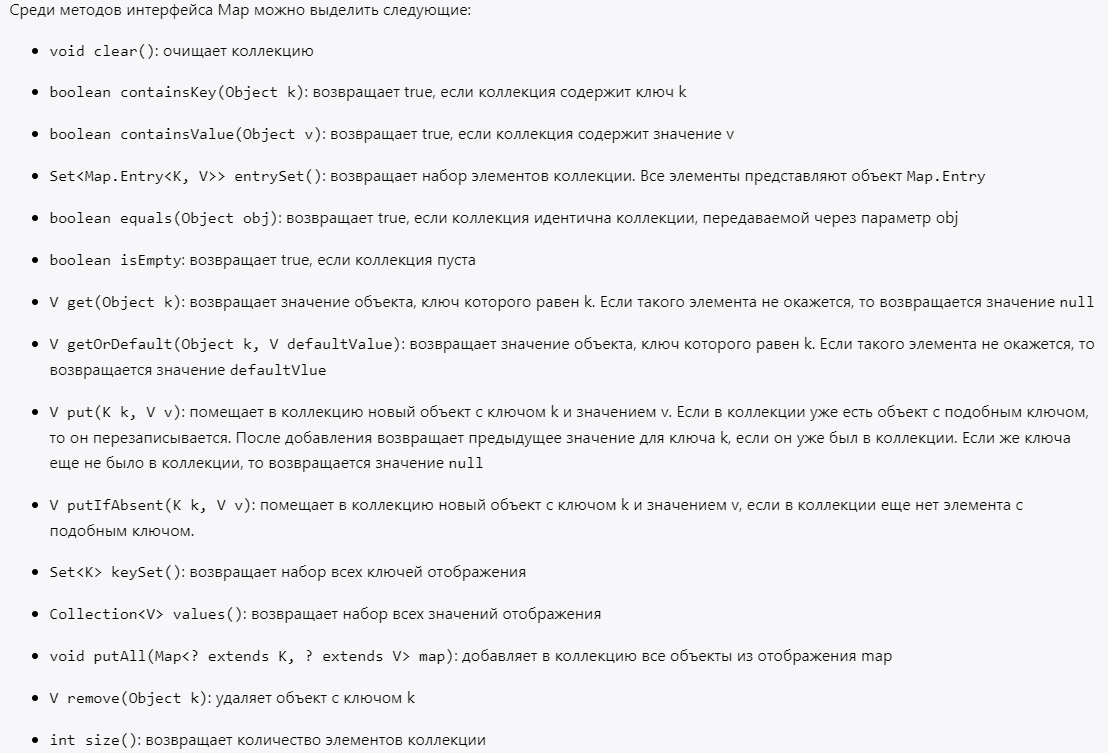
<https://vk.com/marenau>

Интерфейс Map<K, V> представляет отображение или иначе говоря словарь, где каждый элемент представляет пару "ключ-значение". При этом все ключи уникальные в рамках объекта Map. Такие коллекции облегчают поиск элемента, если нам известен ключ - уникальный идентификатор объекта. В отличие от других интерфейсов коллекций не наследуется от интерфейса Collection (!!!).

Базовым классом для всех отображений является абстрактный класс AbstractMap. Наиболее распространенным классом отображений является HashMap, который реализует интерфейс Map и наследуется от класса AbstractMap. HashMap - структура данных в виде словаря, в котором каждый объект имеет уникальный ключ и некоторое значение. В HashMap хеш-таблица реализована на основе динамического массива односвязных списков. По сути, мы получаем хеш-код ключа в результате работы метода hashCode(), который затем модифицируется, а внутри с помощью дополнительного метода полученные значения распределяются по нужным ячейкам. Операции добавления, удаления и поиска будут выполняться за константное время из-за хеширования.

Принцип создания:

Map<Integer, String> states = new HashMap<Integer, String>();



**83.🔒Чем является класс LinkedList<E>**

<https://vk.com/marenau>

LinkedList — класс, реализующий два интерфейса — List и Deque. Это обеспечивает возможность создания списков из любых элементов. Позволяет хранить любые данные, включая null.

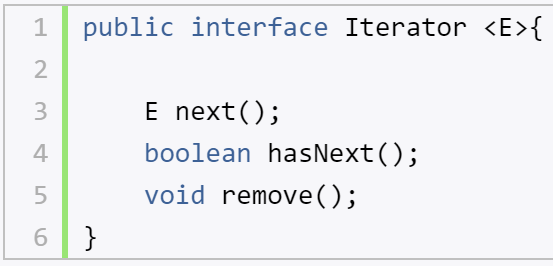
LinkedList -  двусвязный список. Элементы представляют собой звенья одной цепи. У каждого элемента помимо тех данных, которые он хранит, имеется ссылка на предыдущий и следующий элемент. По этим ссылкам можно переходить от одного элемента к другому. Добавление и удаление элемента из середины, доступ по индексу/значению происходят за линейное время, а из начала и конца за константное время.

**84.🔒Одним из ключевых методов интерфейса Collection является метод Iterator<E> iterator(). Что возвращает это метод?**

<https://vk.com/marenau>

Метод Iterator<E> iterator() возвращает итератор - то есть объект, реализующий интерфейс Iterator.

Интерфейс Iterator имеет следующее определение:



Реализация интерфейса предполагает, что с помощью вызова метода next() можно получить следующий элемент. С помощью метода hasNext() можно узнать, есть ли следующий элемент, и не достигнут ли конец коллекции.

**85.🔒Что возвращает метод next()**

<https://vk.com/marenau>

Метод next() возвращает текущий элемент и переходит к следующему, если такого нет, то генерируется исключение NoSuchElementException, поэтому рекомендуется сначала использовать метод проверки - hasNext().

**86. 🔒Что возвращает метод hasNext()**

<https://vk.com/marenau>

Метод hasNext() возвращает true, если в коллекции имеется следующий элемент, иначе возвращает false.

**87.🔒Обобщенный класс HashSet класс коллекция, наследует свой функционал от класса AbstractSet, а также реализует интерфейс Set. Что он себя представляет?**

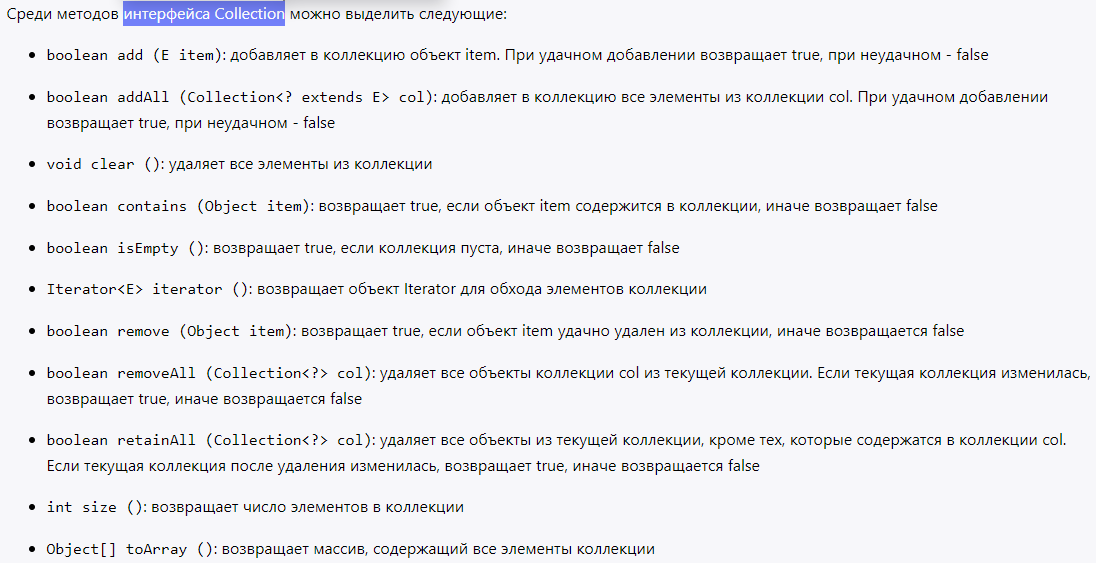
<https://vk.com/marenau>

Интерфейс Set представляет собой неупорядоченную коллекцию, которая не может содержать одинаковые элементы и является программной моделью математического понятия «множество». Интерфейс Set расширяет интерфейс Collection и представляет набор уникальных элементов. Set не добавляет новых методов, только вносит изменения унаследованные. В частности, метод add() добавляет элемент в коллекцию и возвращает true, если в коллекции еще нет такого элемента.

Обобщенный класс HashSet представляет хеш-таблицу. Он наследует свой функционал от класса AbstractSet, а также реализует интерфейс Set. В HashSet элементы не упорядочены, нет никаких гарантий, что элементы будут в том же порядке спустя какое-то время. Операции добавления, удаления и поиска будут выполняться за константное время из-за хеширования.

Принцип создания:

*HashSet<String> states = new HashSet<String>();*



Методы HashSet:

boolean add(E e): добавляет элемент в HashSet, если таковой отсутствует, если же такой элемент уже присутствует, метод возвращает false.

void clear(): удаляет все элементы из множества.

boolean contains(Object o): Возвращает true, если данный элемент присутствует в множестве.

boolean remove(Object o): удаляет данный элемент из множества, если таковой присутствует.

Iterator iterator(): возвращает итератор для элементов множества.

boolean isEmpty(): возвращает true, если в множестве нет элементов.

Object clone(): выполняет поверхностное клонирование HashSet.

**88.🔒Обобщенный класс HashMap класс коллекция, которая реализует интерфейс Map для хранения пар ключ-значение. Что он себя представляет?**

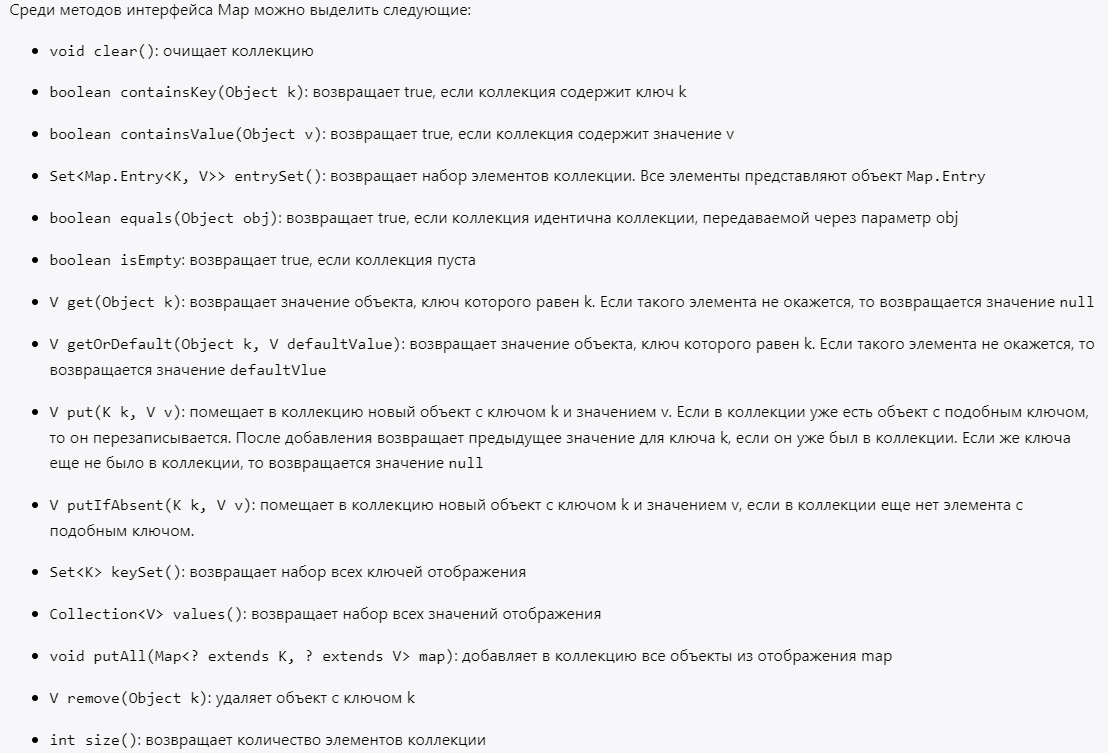
<https://vk.com/marenau>

Интерфейс Map<K, V> представляет отображение или иначе говоря словарь, где каждый элемент представляет пару "ключ-значение". При этом все ключи уникальные в рамках объекта Map. Такие коллекции облегчают поиск элемента, если нам известен ключ - уникальный идентификатор объекта. В отличие от других интерфейсов коллекций не наследуется от интерфейса Collection (!!!).

Базовым классом для всех отображений является абстрактный класс AbstractMap. Наиболее распространенным классом отображений является HashMap, который реализует интерфейс Map и наследуется от класса AbstractMap. HashMap - структура данных в виде словаря, в котором каждый объект имеет уникальный ключ и некоторое значение. В HashMap хеш-таблица реализована на основе динамического массива односвязных списков. По сути, мы получаем хеш-код ключа в результате работы метода hashCode(), который затем модифицируется, а внутри с помощью дополнительного метода полученные значения распределяются по нужным ячейкам. Операции добавления, удаления и поиска будут выполняться за константное время из-за хеширования.

Принцип создания:

*Map<Integer, String> states = new HashMap<Integer, String>();*



**Тема 8. Стандартные потоки ввода-вывода. Сериализация.**

**89.🔒Стандартные потоки ввода-вывода, предоставляемые Java**

два абстрактных класса, представляющие потоки ввода/вывода:

● InPutStream – поток ввода используется для считывания данных с источника.

● OutPutStream – поток вывода используется для записи данных по месту назначения.

Разделяют два вида потоков ввода/вывода: байтовые и символьные.

**90.🔒Понятие сериализации, интерфейс Serializable**

Сериализация представляет процесс записи состояния объекта в поток, соответственно процесс извлечения или восстановления состояния объекта из потока называется десериализацией. Сериализация очень удобна, когда идет работа со сложными объектами.

Интерфейс Serializable

Сразу надо сказать, что сериализовать можно только те объекты, которые реализуют интерфейс Serializable. Этот интерфейс не определяет никаких методов, просто он служит указателем системе, что объект, реализующий его, может быть сериализован.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Сериализация (Serialization) — это процесс, который переводит объект в последовательность байтов, по которой затем его можно полностью восстановить. Зачем это нужно? Дело в том, при обычном выполнении программы максимальный срок жизни любого объекта известен — от запуска программы до ее окончания. Сериализация позволяет расширить эти рамки и «дать жизнь» объекту также между запусками программы.

**91.🔒Какие объекты можно сериализовать?**

Только те объекты, которые реализуют интерфейс Serializable.

**92.🔒Какие методы определяет интерфейс Serializable?**

Этот интерфейс не определяет никаких методов, просто он служит указателем системе, что объект, реализующий его, может быть сериализован.

**93.🔒Что означает понятие десериализация?**

Десериализация — это обратный процесс: восстановление структур и объектов из сериализованной строки или последовательности байтов.

**94.🔒Класс File, определенный в пакете java.io, не работает напрямую с потоками. В чем состоит его задача?**

В отличие от большинства классов ввода/вывода, класс File работает не с потоками, а непосредственно с файлами. Данный класс позволяет получить информацию о файле: права доступа, время и дата создания, путь к каталогу. А также осуществлять навигацию по иерархиям подкаталогов.

Класс java.io.File может представлять имя определённого файла, а также имена группы файлов, находящихся в каталоге. Если класс представляет каталог, то его метод list() возвращает массив строк с именами всех файлов.

У класса очень много методов, перечислим некоторые:

getAbsolutePath() - абсолютный путь файла, начиная с корня системы. В Android корневым элементом является символ слеша (/)

canRead() - доступно для чтения

canWrite() - доступно для записи

exists() - файл существует или нет

getName() - возвращает имя файла

getParent() - возвращает имя родительского каталога

getPath() - путь

lastModified() - дата последнего изменения

isFile() - объект является файлом, а не каталогом

isDirectory - объект является каталогом

isAbsolute() - возвращает true, если файл имеет абсолютный путь

renameTo(File newPath) - переименовывает файл. В параметре указывается имя нового имени файла. Если переименование прошло неудачно, то возвращается false

delete() - удаляет файл. Также можно удалить пустой каталог

**95.🔒При работе с объектом класса FileOutputStream происходит вызов метода FileOutputStream.write(), что в результате этого происходит?**

Класс FileOutputStream предназначен для записи байтов в файл.

Метод write():

write() - записывает один байт в выходной поток файла

write(byte() array) - записывает байты из указанного массива в выходной поток

write(byte() array, int start, int length) - записывает количество байтов, равное длине, в выходной поток из массива, начиная с позиции start