**Список вопросов к экзамену по предмету**

**Программирование на Джава зима 2020-2021 год**

**!!ЖЕЛАТЕЛЬНО ПОДГОТОВИТЬ ФАЙЛ ДО 20 ЯНВАРЯ!!**

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

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

Java – высокоуровневый, объектно-ориентированный язык общего назначения.

Он строго типизирован. (Каждая переменная и константа представляет определенный тип, который строго определен. Тип данных определен диапазоном значений, которые может хранить переменная или константа.)

Java – интерпретируемый язык программирования, т.е. написанный код компилируется в байт-код Java, который передается в интерпретатор (JVM)

**2. Особенности языка Джава**

1. Объектно-ориентированный;
2. Строгая типизация;
3. Встроенная модель безопасности (можно писать многопоточные приложения);
4. Поддерживает многопоточность, динамическая загрузка классов (по необходимости)
5. Синтаксис базируется на языках Cи Си++
6. Платформонезависимость (архитектурно-нейтральный) - независимость от платформы (системы), для исполнения программы (благодаря Java Virtual Machine (JVM))
7. Необходимость присутствия в платформе (системе) пользователя интерпретатора (Java Virtual Machine (JVM))

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

***Scanner*** – это класс, который позволяет считывать данные из входящих потоков. В том числе и с консоли.

Класс Scanner часть библиотеки java.util и должен быть импортирован в программу, чтобы можно было им пользоваться.

В конструкторе класса можно указывать поток ввода (например, стандартный или открытого файла), путь к файлу или строку.

***Ввод с клавиатуры*** представлен объектом System.in

Следующая строка создает объект Scanner, который считывает с клавиатуры:

Scanner scan = new Scanner (System.in);

Метод nextLine считывает весь ввод до конца строки.

Метод next класса Scanner читает следующий элемент на входе и возвращает его в виде строки. Метод hasNext[название\_типа](), проверяет, можно ли считать с потока ввода символ соответствующего типа, а метод next[название\_типа]() — считывает его.

Если попытаться считать значение без предварительной проверки, то во время исполнения программы можно получить ошибку (отладчик заранее такую ошибку не обнаружит).

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

***Scanner*** – это класс, который позволяет считывать данные из входящих потоков. В том числе и с консоли.

Класс Scanner часть библиотеки java.util и должен быть импортирован в программу, чтобы можно было им пользоваться.

В конструкторе класса можно указывать поток ввода (например, стандартный или открытого файла), путь к файлу или строку.

***Поток стандартного ввод (с клавиатуры)*** представлен объектом System.in

Следующая строка создает объект Scanner, который считывает со стандартного потока ввода:

Scanner scan = new Scanner (System.in);

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

**nextLine()**

Метод **nextLine** считывает весь ввод до конца строки.

------------------------------------------------------------------

import java.util.Scanner;

public class Main{

public static void main (String[] args){

String message; Scanner scan = new Scanner (System.in); System.out.println ("Enter a line of text:");

message = scan.nextLine();

System.out.println ("You entered: \"" + message + "\"");

}

}

------------------------------------------------------------------

**nextInt()**

Метод **next** класса **Scanner** читает следующий элемент на входе и возвращает его в виде строки. Такие методы как **nextInt** и **nextDouble** считывают данные конкретных типов данных.

**hasNextInt()**

Метод **hasNextInt()**, применённый к объекту класса Scanner, проверяет, можно ли считать с потока ввода целочисленное значение типа **int**, а метод **nextInt()** — считывает его. Если попытаться считать значение без предварительной проверки, то во время исполнения программы можно получить ошибку (отладчик заранее такую ошибку не обнаружит).

**hasNextLine()**

Существуют методы **hasNext\_название\_типа**() для различных примитивных типов. Эти методы перегружены для работы с различными базовыми типами данных. Например, для типа **double** будет использоваться метод **hasNextDouble()** и метод **nextDouble ()**.

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

**Целочисленные**

1. char – 16-битовый символ Unicode,

2. byte – 8-битовое целое число со знаком,

3. short – 16-битовое целое число со знаком,

4. int – 32-битовое целое число со знаком,

5. long – 64-битовое целое число со знаком.

**Вещественные типы:**

6. float – 32-битовое число с плавающей точкой (IEEE 754-1985),

7. double – 64-битовое число с плавающей точкой (IEEE 754-1985).

**Логические:**

8. boolean допускает хранение значений true или false.

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

**Неполное ветвление -**Условный оператор if позволяет задать условие, в соответствии с которым дальнейшая часть программы может быть выполнена.

------------------------------------------------------------------

int a = 20;

if(a == 20){

System.out.println("a = 20");

}

------------------------------------------------------------------

**Полное -** Условный оператор if-else в языке означает «в ином случае». То есть если условие if не является истинным, выводим то, что в блоке else.

------------------------------------------------------------------

int a = 20; int b = 5;

if(a == 20){

System.out.println("a = 20");

}

else(b == 5){

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

}

------------------------------------------------------------------

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

Условный оператор или оператор множества switch позволяет сравнить переменную как с одним, так и с несколькими значениями.

Пример:

------------------------------------------------------------------

switch(выражение) {

case значение1:

// Блок кода 1

break;

case значение2:

// Блок кода 2

break;

case значениеN:

// Блок кода N

break;

default :

// Блок кода для default

}

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

Системный является одним из базовых классов в Java и принадлежит пакету java.lang. Класс System является финальным и не предоставляет общедоступных конструкторов. Из-за этого все члены и методы, содержащиеся в этом классе, являются статическими по природе.

Поскольку класс System имеет множество ограничений, существуют различные предварительно созданные поля и методы класса. Ниже я перечислил несколько важных функций, поддерживаемых этим классом:

Стандартный ввод и вывод.

Ошибка вывода потоков.

Доступ к внешним свойствам и переменным среды.

Встроенная утилита для копирования части массива.

Предоставляет средства для загрузки файлов и библиотек.

1. Объявление: public final class System extends Object
2. Поля класса: Класс java.lang.System поставляется с тремя полями:

*public static final InputStream in* - это стандартный поток ввода в Java-программировании. Этот поток уже открыт и доступен для ввода входных данных.

*public static final PrintStream out* - это стандартный поток вывода в Java-программировании .

*public static final PrintStream err* - это стандартный поток вывода ошибок в Java-программировании. Технически этот выходной поток используется для отображения сообщений об ошибках или другой информации, которая требует немедленного внимания пользователя.

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

Класс OutputStream является классом-родителем для всех классов, которые поддерживают байтовый вывод. Это абстрактный класс, который сам ничего не делает: для этого у него есть классы-наследники на все случаи жизни.

Если попроще, этот класс оперирует байтами, а не, например, символами или другими типами данных. А то, что он абстрактный, значит, что мы обычно используем не его, а один из его классов-наследников. Например, FileOutputStream и ему подобные.

Вернемся к классу OutputStream. У этого класса есть методы, которые обязаны реализовывать все его классы-наследники. Вот основные из них:

| **Методы** | **Описание** |
| --- | --- |
| void write(int b) | Записывает один байт (не int) в поток. |
| void write(byte[] buffer) | Записывает массив байт в поток |
| void write(byte[] buffer, off, len) | Записывает часть массива байт в поток |
| void flush() | Записывает в поток все данные, которые хранятся в буфере |
| void close() | Закрывает поток |

При создании объекта класса-наследника InputStream обычно указывается объект-источник, из которого InputStream читает данные. При создании объекта класса-наследника OutputStream также обычно указывается целевой объект или целевой поток, в который будут записываться данные.

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

Константы в Java используются, когда необходимо реализовать «статическое» или постоянное значение для переменной. Язык программирования напрямую не поддерживает константы. Чтобы сделать любую переменную ею, мы должны использовать модификаторы static и final.

Синтаксис:*static final datatype identifier\_name = constant;*

Модификатор static делает переменную доступной без загрузки экземпляра ее определяющего класса. Последний модификатор делает переменную неизменной.

Причина, по которой мы должны использовать как статические, так и конечные модификаторы, заключается в том, что:

1. Когда мы объявим переменную «var» только как статическую, все объекты одного класса смогут получить доступ к этому ‘var’ и изменить его значения.
2. Когда мы объявляем переменную только как final, для каждого отдельного объекта будет создано несколько экземпляров одного и того же значения константы, и это неэффективно / нежелательно.
3. Когда мы используем как static, так и final, тогда «var» остается статичным и может быть инициализирован только один раз, что делает его надлежащей константой, которая имеет общую ячейку памяти для всех объектов своего содержащего класса.

**12. В результате выполнения этой строчки**

?

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

Для объявления бестиповых данных используется ключевое слово var. Компилятор сам подбирает тип переменных, опираясь на присвоенные им значения. При объявлении такой переменной сразу должна быть инициализация, иначе будет ошибка.

var используется в объявлении локальной переменной без явного указания типа. При использовании var компилятор Java определяет тип переменной во время компиляции, используя информацию о типе, полученную от инициализатора переменной. Далее выведенный тип используется как статический тип этой переменной.

var person = new Person();

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

Ссылочные типы данных. **Ссылочными типы** — это переменные типа класс, которые используются для ссылок на объекты. **Ссылочные переменные** хранят адрес ячейки памяти, в которой расположен определенный объект. Ссылочные переменные создаются с использованием определенных конструкторов классов.

Объявление переменной типа класс в Java:

MyClass1 obj1;

MyClass2 obj1;

Примеры создания объектов:

obj1 = new MyClass1 ();

obj2 = new MyClass2 (12, 5);

Для создания объекта obj1 вызывается конструктор класса MyClass1 без параметров, а для создания объекта obj2 вызывается конструктор класса MyClass2 с параметрами, которому в качестве аргумента передаются два целых числа.

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

**Арифметические операции:**

«+» операция сложения двух чисел: int a = 3 + 2;

«-» операция вычитания двух чисел: int b = 3 - 2;

«\*» операция умножения двух чисел: int c = 3 \* 2;

«/» операция деления двух чисел: double d = 3 / 2;

«%» получение остатка от деления нацело двух чисел: int e = 3%2;

**Операции инкремента:**

«++» (префиксный инкремент)

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

«++» (постфиксный инкремент)

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

**Операции декремента:**

«--» (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

«--» (постфиксный декремент)

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

int a = 8;

int b = a--;

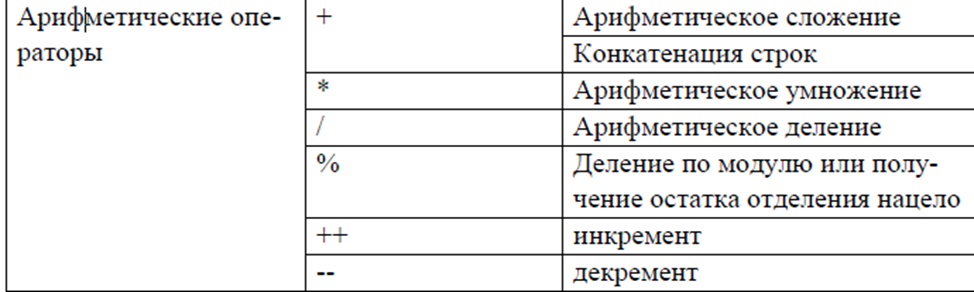
System.out.println(a); // 7

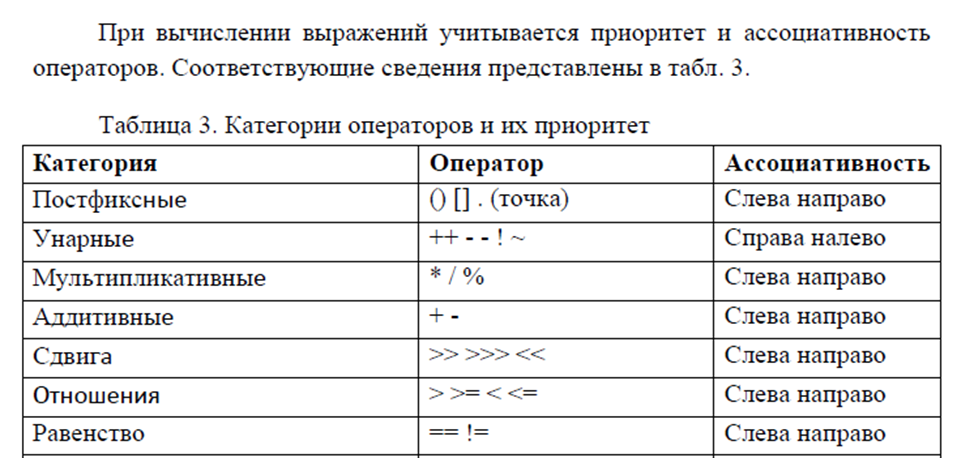
System.out.println(b); // 8

**16. В результате выполнения фрагмента программы**

?

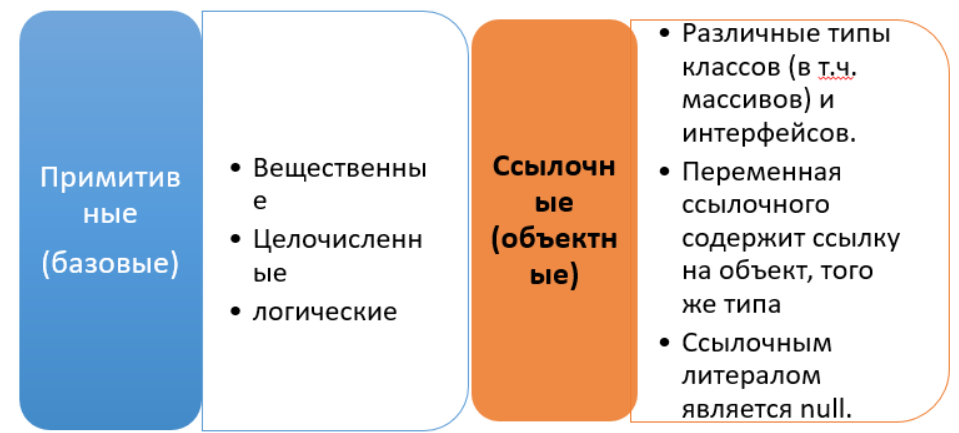
**17. Арифметические операции, приоритет выполнения операций**

****

****

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

Все типы данных в языке Джава можно разделить на две группы. Первую группу представляют примитивные типы данных – числовые типы и логический тип. Вторую группу данных представляют объектные типы данных. На рис. 1.18 схематично изображены типы данных языка Джава.



***Примитивные (простые) типы данных***

*К простым типам* данных относятся типы для представления числовых значений и также логический тип

К целочисленным типам относятся типы:

1) char – 16-битовый символ Unicode,

2) byte – 8-битовое целое число со знаком,

3) short – 16-битовое целое число со знаком,

4) int – 32-битовое целое число со знаком,

5) long – 64-битовое целое число со знаком.

***Вещественные типы данных***

К вещественным типам относятся типы:

1. float – 32-битовое число с плавающей точкой (IEEE 754-1985),

2. double – 64-битовое число с плавающей точкой (IEEE 754-1985).

***Логический тип***

*Логический тип* boolean допускает хранение значений true или false.

Листинг программы 1.12 – Базовые типы

public class Example5{

public static void main(String[] args){

short x = 10, y = 5, z=1; byte B = 17;

long w = 234334000000L; double d = 5.01;

float f = 5.01f;

char ch = 'a';

String s = "Hello World!"; //строка

boolean isFlag = false;

System.out.println("ch = " + ch );

System.out.println("isGlag = " + isFlag);

System.out.println(s);

}

}

***Ссылочные типы данных***

Ссылочными типы — это переменные типа класс, которые используются для ссылок на объекты.

К объектным типам относят:

1) Классы – переменные типа класс

2) Интерфейсы – ссылки интерфейсного типа

3) Перечисления –поименованные константы

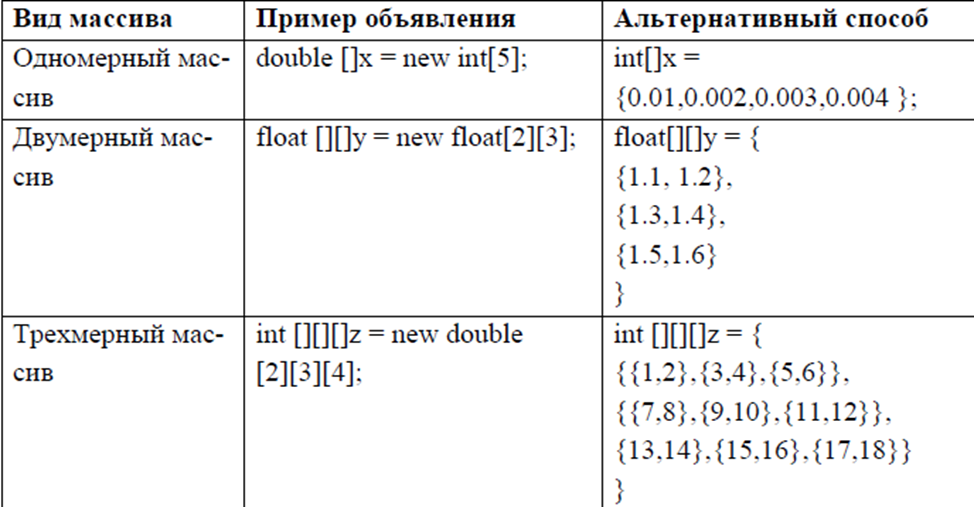
4) Типы оболочки соответствуют каждому примитивному типу

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

Массив является фундаментальной структурой для хранения данных в программах. Особенность. Вы можете в массивах Джава хранить различные типы данных, это могут быть как базовые, например примитивные или объектные типы данных. Также вы можете хранить в качестве элементов массивов свои собственные типы данных. В Джава есть несколько альтернативных способов объявления и инициализации массивов.

***Объявление и инициализация массивов***

Объявление массивов несколько отличается от языка С/С++. Сначала указывается тип элементов массива, потом квадратные скобки – синтаксис массива и затем имя массива.



Инициализировать массив можно точно таким же способом, как в языке Си, с помощью прямой инициализации, не указывая размерность массива.

Например, как представлено ниже:

int[] sample = {12, 56, 7, 34, 89, 43, 23, 9};

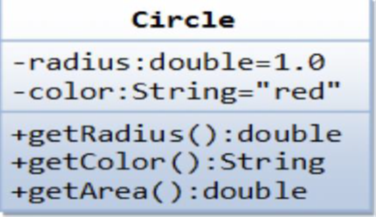
Или другим способом, если предполагается хранить в качестве элементов массива объекты, например, как представлено ниже:

Circle[] array = { new Circle(1, 1, “red”), new Circle(3, 4, “green”), new Circle(1, 3, “”)};

Массив в Java это объектный тип данных, соответственно у него есть некоторые поля и методы.

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

Массив в Java это объектный тип данных, соответственно у него есть некоторые поля и методы.



Если предполагается хранить в качестве элементов массива объекты, например, как представлено ниже:

Circle[] array = { new Circle(1, 1, “red”), new Circle(3, 4, “green”), new Circle(1, 3, “”)};

Листинг программы 1.15 – Пример 1 работы с массивом

public class Main{

public static void main(String[] args) {

// Объявление массива и его создание

Circle[] circles = new Circle[10];

}

}

Программа на листинге 1.15 в цикле печатает элементы массива.

Листинг программы 1.15 – Пример 2 работы массивом

public class Main {

public static void main(String[] args) {

// Объявление массива и создание

Circle[] circles = new Circle[10];

/\* С помощью цикла изменяем переменную i и ис-пользуем ее\*/

// для обращения к элементу массива

for(int i=0; i<10; i++) {

System.out.println(circles[i]);

// Печатаем элемент массива

}

}

}

Программа на листинге 1.16 печатает объекты - элементы массива благо-даря методу toString(), переопределенному в классе Circle.

Листинг программы 1.16 – Пример 3 с массивом

public class Main {

public static void main(String[] args) {

// Объявление массива и создание

Circle[] circles = new Circle[10];

// Еще один цикл, который вызывает печать

for (int i = 0; i < circles.length; i++) {

circles[i].toString();

}

}

}

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

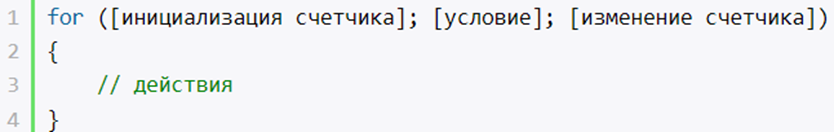
**22. В результате выполнения фрагмента программы**

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

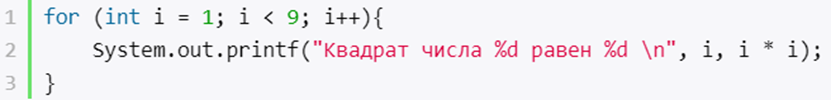
**24. Циклы в Джава, цикл с предусловием, цикл с постусловием, пример записи и использование. Условие окончания цикла.**

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

Цикл for имеет следующее формальное определение:



Рассмотрим стандартный цикл for:



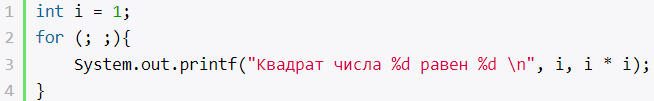
Первая часть объявления цикла - int i = 1 создает и инициализирует счетчик i. Счетчик необязательно должен представлять тип int. Это может быть и любой другой числовой тип, например, float. Перед выполнением цикла значение счетчика будет равно 1. В данном случае это то же самое, что и объявление переменной.

Вторая часть - условие, при котором будет выполняться цикл. В данном случае цикл будет выполняться, пока i не достигнет 9.

И третья часть - приращение счетчика на единицу. Опять же нам необязательно увеличивать на единицу. Можно уменьшать: i--.

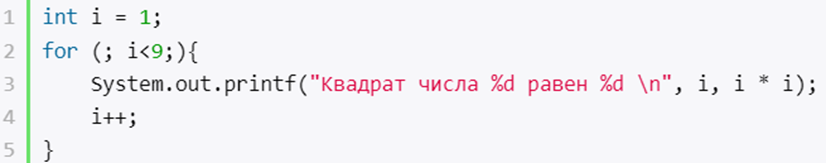
В итоге блок цикла сработает 8 раз, пока значение i не станет равным 9. И каждый раз это значение будет увеличиваться на 1.

Нам необязательно указывать все условия при объявлении цикла. Например, мы можем написать так:



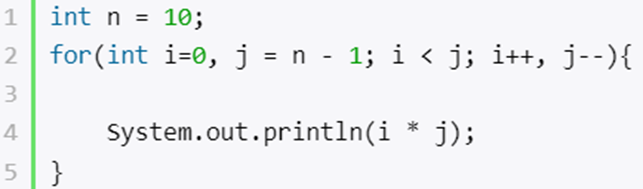
Определение цикла осталось тем же, только теперь блоки в определении у нас пустые: for (; ;). Теперь нет инициализированной переменной-счетчика, нет условия, поэтому цикл будет работать вечно - бесконечный цикл.

Либо можно опустить ряд блоков:



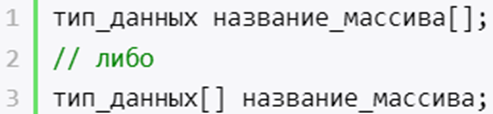
Этот пример эквивалентен первому примеру: у нас также есть счетчик, только создан он вне цикла. У нас есть условие выполнения цикла. И есть приращение счетчика уже в самом блоке for.

Цикл for может определять сразу несколько переменных и управлять ими:

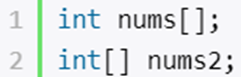


**26. Способы объявления массивов в Джава, использование операции new для выделения памяти для элементов массива. Объявление с инициализацией, объявление массива определенного размера без инициализации.**

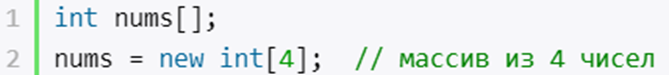
Массив представляет набор однотипных значений. Объявление массива похоже на объявление обычной переменной, которая хранит одиночное значение, причем есть два способа объявления массива:



Например, определим массив чисел:

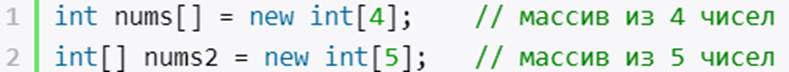


После объявления массива мы можем инициализировать его:



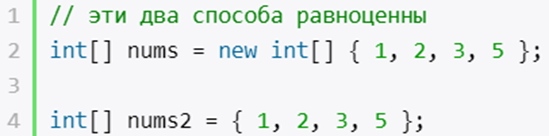
Создание массива производится с помощью следующей конструкции: **new тип\_данных[количество\_элементов]**, где new - ключевое слово, выделяющее память для указанного в скобках количества элементов. Например, nums = new int[4]; - в этом выражении создается массив из четырех элементов int, и каждый элемент будет иметь значение по умолчанию - число 0.

Также можно сразу при объявлении массива инициализировать его:



При подобной инициализации все элементы массива имеют значение по умолчанию. Для числовых типов (в том числе для типа char) это число 0, для типа boolean это значение false, а для остальных объектов это значение null. Например, для типа int значением по умолчанию является число 0, поэтому выше определенный массив nums будет состоять из четырех нулей.

Однако также можно задать конкретные значения для элементов массива при его создании:



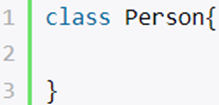
Стоит отметить, что в этом случае в квадратных скобках не указывается размер массива, так как он вычисляется по количеству элементов в фигурных скобках.

**Тема 2. Реализация ООП в Java.**

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

Шаблоном или описанием объекта является класс, а объект представляет экземпляр этого класса. Можно еще провести следующую аналогию. У нас у всех есть некоторое представление о человеке - наличие двух рук, двух ног, головы, туловища и т.д. Есть некоторый шаблон - этот шаблон можно назвать классом. Реально же существующий человек (фактически экземпляр данного класса) является объектом этого класса.

Класс определяется с помощью ключевого слова сlass:



В данном случае класс называется Person. После названия класса идут фигурные скобки, между которыми помещается тело класса - то есть его поля и методы.

Любой объект может обладать двумя основными характеристиками: состояние - некоторые данные, которые хранит объект, и поведение - действия, которые может совершать объект.

Для хранения состояния объекта в классе применяются поля или переменные класса. Для определения поведения объекта в классе применяются методы. Например, класс Person, который представляет человека, мог бы иметь следующее определение:



**28. Использования this для доступа к компонентам класса.**

Ключевое слово this представляет ссылку на текущий экземпляр класса. Через это ключевое слово мы можем обращаться к переменным, методам объекта, а также вызывать его конструкторы. Например:

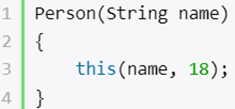


В третьем конструкторе параметры называются так же, как и поля класса. И чтобы разграничить поля и параметры, применяется ключевое слово this:



Так, в данном случае указываем, что значение параметра name присваивается полю name.

Кроме того, у нас три конструктора, которые выполняют идентичные действия: устанавливают поля name и age. Чтобы избежать повторов, с помощью this можно вызвать один из конструкторов класса и передать для его параметров необходимые значения:



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

Создание объекта — это процесс выделения памяти для хранения данных в полях класса (также называемых переменными). Этот процесс часто называют созданием экземпляра класса.

Ключевое слово new

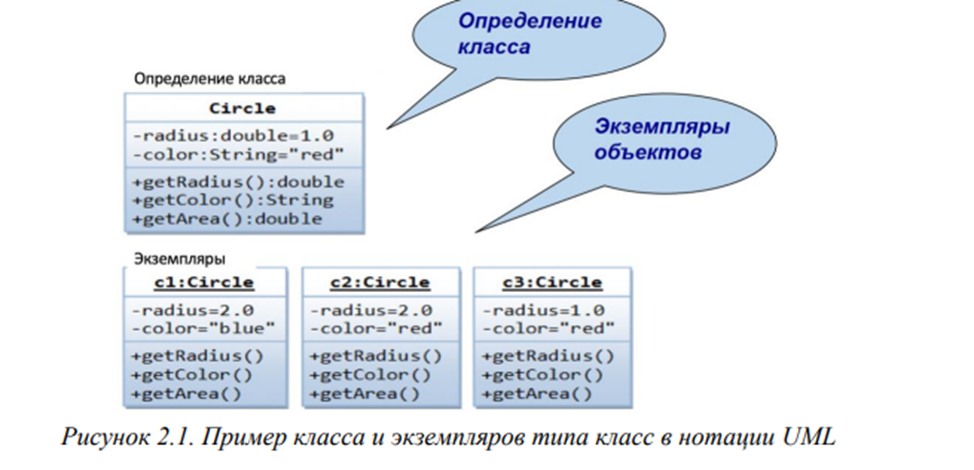
Оператор new возвращает ссылку на объект, размещенный памяти. Как работает new? Сначала выделяется память, а потом выполняется присваивание. Фактически это две разных операции. Создание объектов называется инстанцированием. Любой объект - экземпляр типа класс (instance of class).

После new идет конструктор — специальный метод, отвечающий за создание объекта и инициализацию полей создаваемого объекта.

Пример:



**30. Что такое класс в Java?**

Классы — это типы данных, создаваемые программистом. На этапе проектирования программы после сбора и анализа требований определяются сущности, которые могут служить в качестве «кандидатов» для разработки класса. Классы проектируется на этапе проектирования программы. Проектирование программы выполняют аналитики после сбора и анализа требований в виде UML диаграмм. Отношения между классами также определяются на этапе проектирования программы. Это могут отношения композиции, агрегации, наследования или ассоциации. А также использования. Примеры нотации UML для описания класса и экземпляров типа класс представлены на рис. 2.1. (Все это из лекции в сдо)

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

В Джава существуют различные виды модификаторов. Рассмотрим вначале модификаторы видимости. Их еще называют модификаторы доступа. Роль модификаторов видимости в программах на Джава обеспечить область видимости для данного компонента класса, в зависимости от того с каким модификатором он объявлен.

Всего в Java есть четыре модификатора доступа. Перечислим их в порядке от самых строгих до самых «мягких»:

* **Private** — наиболее строгий модификатор доступа. Он ограничивает видимость данных и методов пределами одного класса.**;**
* **Protected** —Поля и методы будут видны в пределах всех классов, находящихся в том же **пакете, что и наш**;в пределах **всех классов-наследников** нашего класса.
* **default (package visible)** —Переменные или методы с таким модификатором (т.е. вообще без какого-нибудь) видны всем классам пакета, в котором они объявлены. И только им.**;**
* **public** — Части кода, помеченные модификатором public, предназначаются для конечного пользователя и доступны в любом месте программы.

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

Кроме статических переменных, в классах могут быть и статические методы.

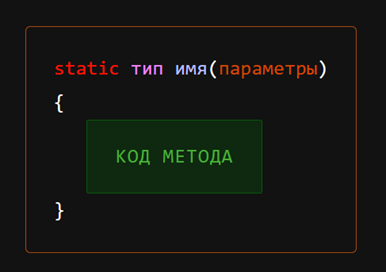
Обычные методы привязаны к объектам (экземплярам) класса и могут обращаться к обычным-переменным класса (а также к статическим переменным и методам). Статические же методы привязаны к статическому объекту класса и могут обращаться только к статическим переменным и/или другим статическим методам класса.

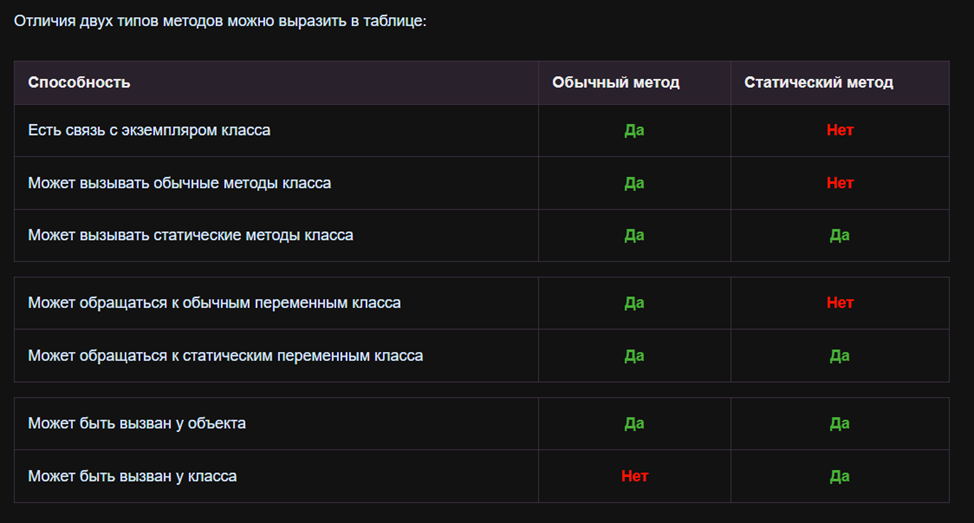
Чтобы вызвать обычный метод у класса, сначала нужно создать объект этого класса, а только потом вызвать метод у объекта. Вызвать обычный метод не у объекта, а у класса нельзя.

А чтобы вызвать статический метод, достаточно чтобы просто существовал статический объект класса (который всегда существует после загрузки класса в память). Именно поэтому метод main() — статический. Он привязан к статическому объекту класса, для его вызова не нужно создавать никакие объекты.

Чтобы объявить метод статическим, нужно перед заголовком метода написать ключевое слово static.

Общий вид этой конструкции такой:





Зачем такие методы нужны, если они так сильно ограничены? Все дело в том, что у такого подхода тоже есть свои преимущества.

Во-первых, чтобы обратиться к статическим методам и переменным, не надо передавать никакую ссылку на объект.

Во-вторых, иногда бывает нужно, чтобы переменная была в единственном экземпляре. Как, например, переменная System.out (статическая переменная out класса System).

И, в-третьих, иногда нужно вызвать метод еще до того, как будет возможность создавать какие-то объекты. Например, вызов метода main(), с которого начинается выполнение программы: его вызывает Java-машина до создания экземпляра класса.

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

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

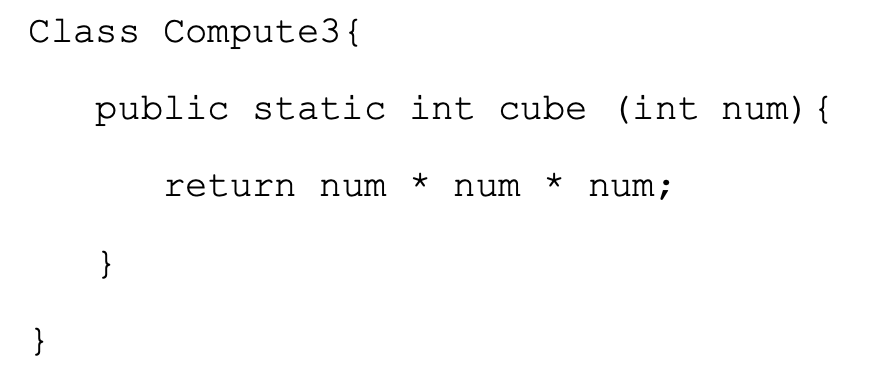
**35. Как вызвать обычный метод класса внутри static-метода?**

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

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

Модификатор static Мы объявляем статические методы и переменные, используя модификатор static. Это связывает метод или переменную с классом, а не с объектом этого класса. Статические методы иногда называют методами класса и статические переменные иногда называют переменные класса

**Статические методы**

На листинге 3.1 представлена класс Compute3 в котором функция есть статическая функция cube. 

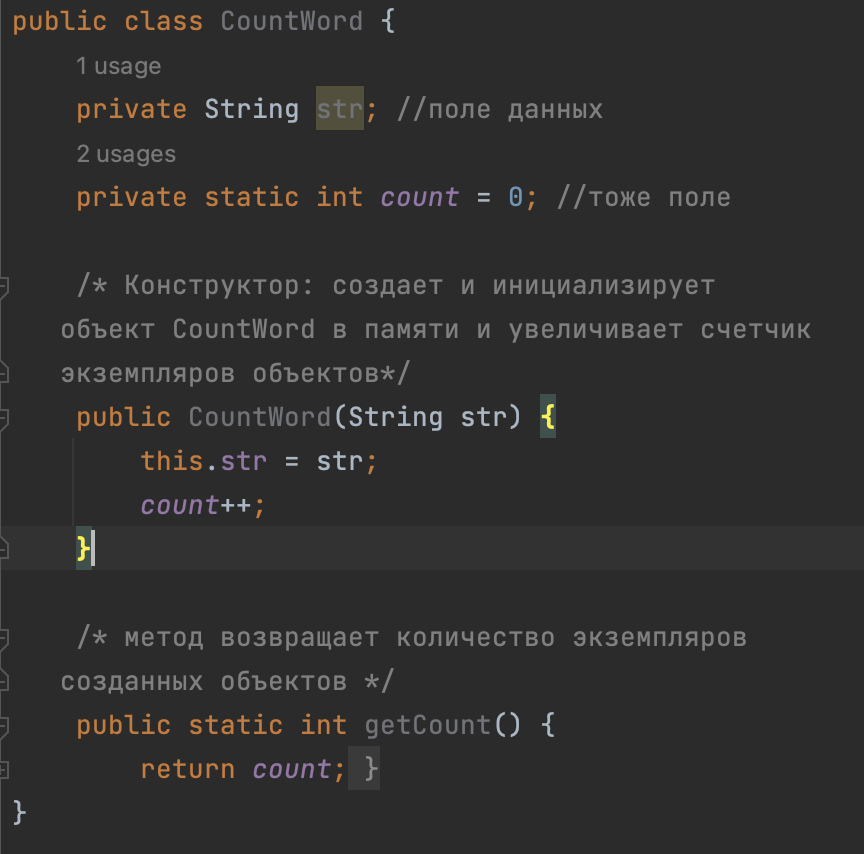
Поскольку функция cube() объявлена как статическая, то молжно вызвать ее не создавая экземпляра объекта типа класс Compute3, вот таким образом:

value = Compute3.cube(5);

Статические методы не могут ссылаться на переменные экземпляра, поскольку пока объект не существует, то переменных экземпляра тоже не существует. Тем не менее, статический метод может ссылаться на статические переменные или локальные переменные.

Статические методы и статические переменные часто работают вместе.

Следующий пример класса CountWord позволяет отследить, сколько объектов будет создано с помощью статической переменной, и эта информацию будет доступна с помощью статического метода.



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

**39. Методы с пустым списком параметров**

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

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

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

**43. Может ли класс иметь в своем составе несколько конструкторов? 44. Может ли конструктор класса возвращать значение?**

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

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

Наследование классов в Java реализуется как одиночное наследование. Множественное наследование реализуется только через интерфейсы. В Java, мы используем служебное слово extends - расширяет для определения классом отношения наследования.

Например, в коде, приведенном ниже класс CheckingAccount расширяет класс BankAccount.

public class CheckingAccount extends BankAccount {

/\*\*тело класса\*/

}

При наследовании все объекты нового класса получают или наследуют от родителя все поля (состояния) и методы (поведение) родительского класса. По правилам Джава конструкторы и статические методы или статические поля не наследуются. По умолчанию, родительский класс для всех классов, которые вы создаете это класс Object. Сила Java в том, что только один родительский класс (“единичное наследование”). То есть в языке Джава нет возможностей организовать, например ромбовидное наследование как в языке С++. У каждого класса всегда есть только один родитель, но несколько классов с помощью наследования могут организовывать цепочку наследования.

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

В программе шмы можем использовать методы с одним и тем же именем, но с разными типами и/или количеством параметров. Такой механизм называется перегрузкой методов (method overloading).

Например:

public class Program{

public static void main(String[] args) {

System.out.println(sum(2, 3)); // 5

System.out.println(sum(4.5, 3.2)); // 7.7

System.out.println(sum(4, 3, 7)); // 14

}

static int sum(int x, int y){

return x + y;

}

static double sum(double x, double y){

return x + y;

}

static int sum(int x, int y, int z){

return x + y + z;

}

}

Здесь определено три варианта или три перегрузки метода sum(), но при его вызове в зависимости от типа и количества передаваемых параметров система выберет именно ту версию, которая наиболее подходит.

Стоит отметить, что на перегрузку методов влияют количество и типы параметров. Однако различие в типе возвращаемого значения для перегрузки не имеют никакого значения.

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

В момент наследования - может возникнуть **проблема**. Чтобы проиллюстрировать, представим, что у нас есть класс **Animal**. Этот класс имеет несколько методов - в том числе и метод **voice()** ("голос"):

| 1  2  3  4  5  6 | public class Animal {  // ...  void voice() {  System.out.println("This is my voice!");  }  } |
| --- | --- |

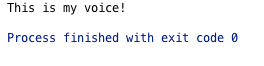
Давайте теперь представим, что мы создаем класс **Cat**, который наследует **Animal**:

| 1  2  3 | public class Cat extends Animal {  // ...  } |
| --- | --- |

Но сейчас если мы создадим кошку и запустим метод **voice()** , мы получим **"This is my voice!"**. Например, если запустим это:

| 1  2  3  4  5  6  7  8 | public class Test {    public static void main(String[] args) {  Cat MyCat = new Cat();    MyCat.voice();  }  } |
| --- | --- |

Получаем:



Как мы знаем, кошка не говорит "**This is my voice!**" Кошка мяукает, собака лает, и т.д.

Таким образом, мы хотели бы, чтобы при вызове метода **voice()** наша кошка говорила "Мяу". Конечно, мы можем создать новый метод - например, **catVoice()**, - но это было бы не очень эффективно. Что, если мы хотим изменить 3, 5 или 10 методов? И что, метод родителя будет лежать у нас мертвым грузом?

Нет, нам совершенно не обязательно идти этим путем. Гораздо лучше, если мы просто будем иметь **"свой вариант"** нужного метода. Мы можем сделать так, чтобы нам не надо было менять название или параметры метода, но он вызвал нужный нам результат. Для этого мы просто должны **переопределить** **метод** родителя.

**Пример**

Как нам переопределить метод **voice()**? Все очень просто:

* Во-первых, мы заново прописываем метод **voice()** в нашем классе **Cat**
* Во-вторых, мы должны дать Java понять, что мы не ошиблись и не назвали метод названием, которое уже есть, из-за невнимательности. Чтобы сказать, что мы знаем, что делаем и действительно хотим **переопределить** **(**по англ. "**override**"**)** метод, нужно над методом написать **@Override**.

В коде, например, это выглядит так:

| 1  2  3  4  5  6  7 | public class Cat extends Animal {    @Override  void voice() {  System.out.println("Meow");    } |
| --- | --- |

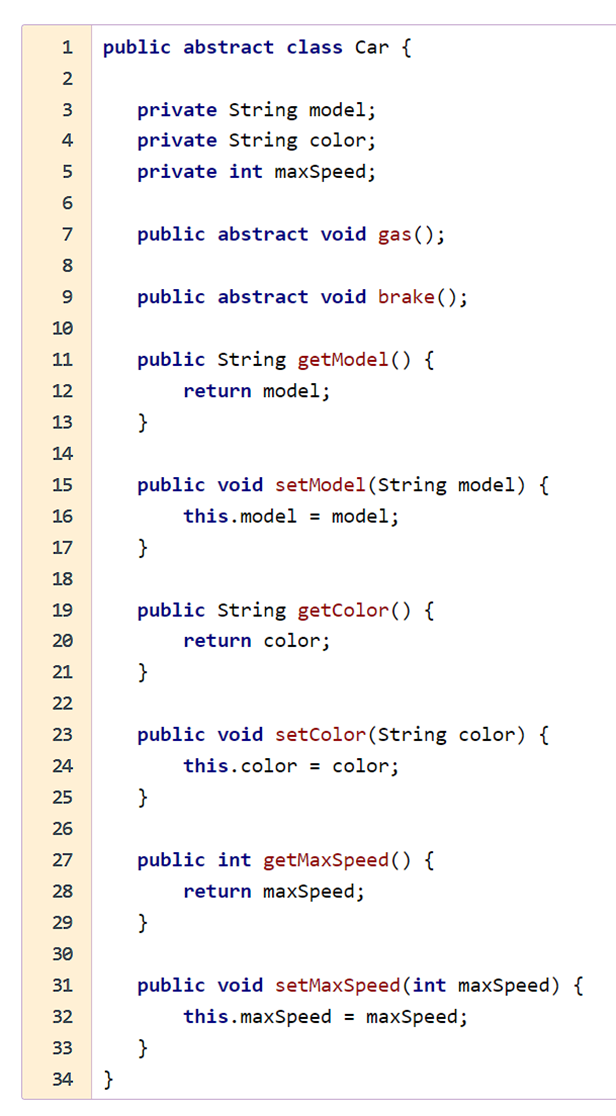
**48. В чем разница между перегрузкой и переопределением методов, поясните**

Переопределением называют случаи, когда сигнатура метода (имя и параметры) в суперклассе и дочернем классе совпадают. Когда два или более метода в одном классе имеют одинаковое имя, но разные параметры, это называется перегрузкой.

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

**Абстрактный класс** – это тип класса, который содержит наиболее общие (абстрактные) параметры. Это “заготовка” для группы будущих классов, которая описывает лишь общее их поведение. **Пример – мебель или машина**. Это лишь группы, но не единичные экземпляры.

Синтаксис абстрактного класса:

****

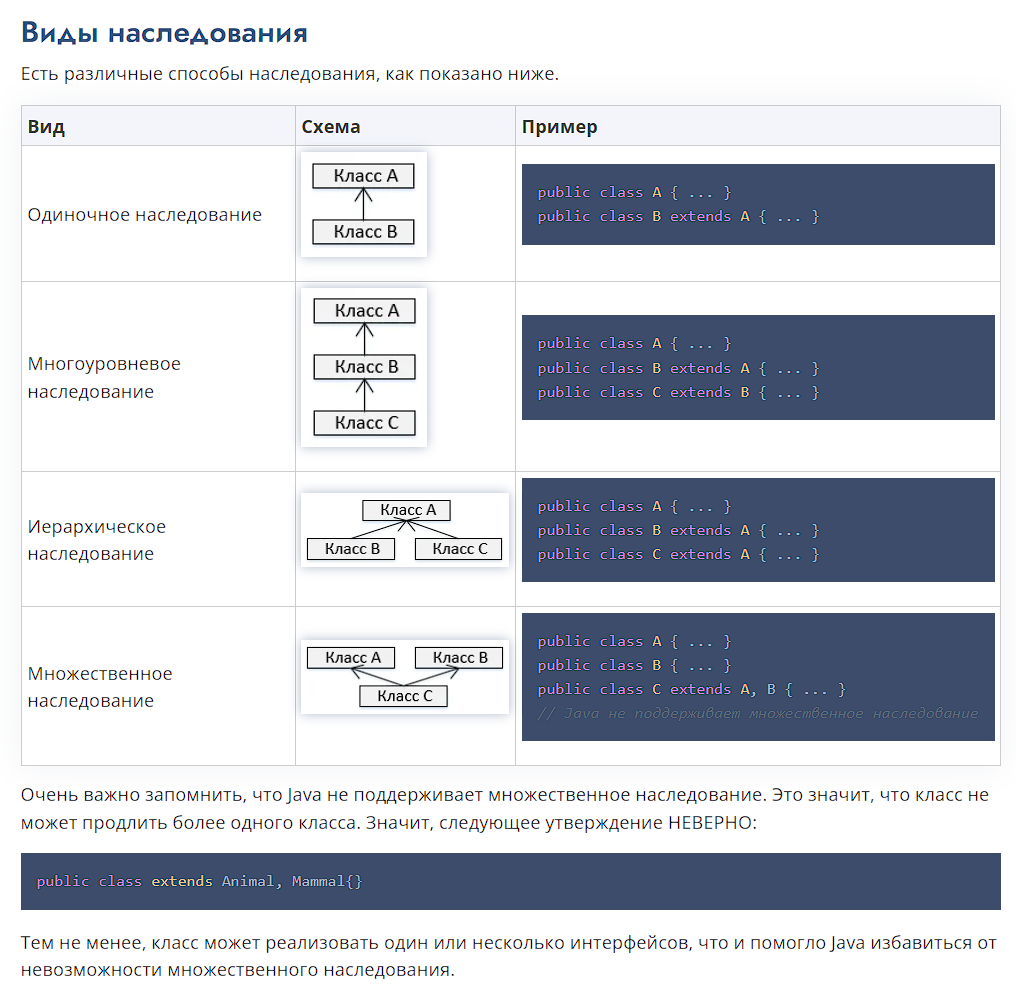
Наследник абстрактного класса:

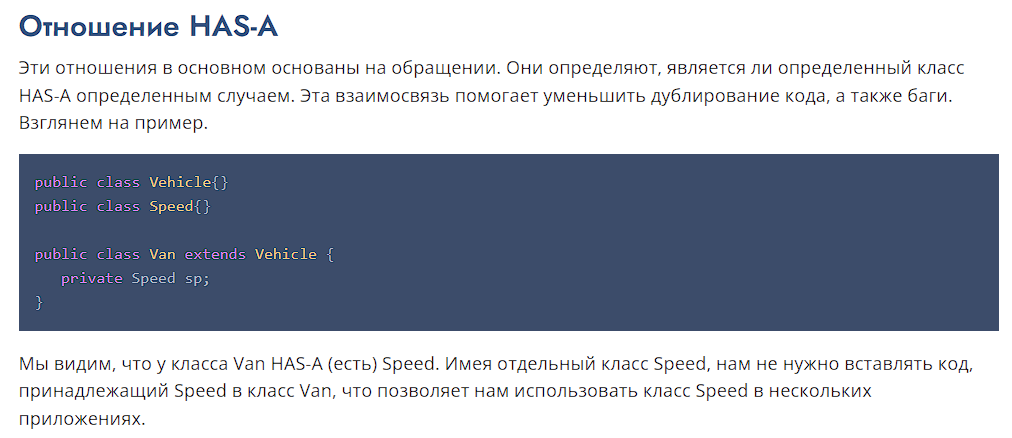


Важные особенности:

* Экземпляр абстрактного класса создать нельзя!
* В классе наследнике обязательно надо переопределить все абстрактные методы, иначе этот класс также будет необходимо сделать абстрактным.
* Абстрактные методы в абстрактном классе можно как реализовать, так и оставить пустыми. Это не вызовет ошибки компилятора, но по сути логика абстрактности исчезнет при начальной реализации и так делать не принято.
* Класс абстрактный, если хотя бы один из его методов абстрактный.

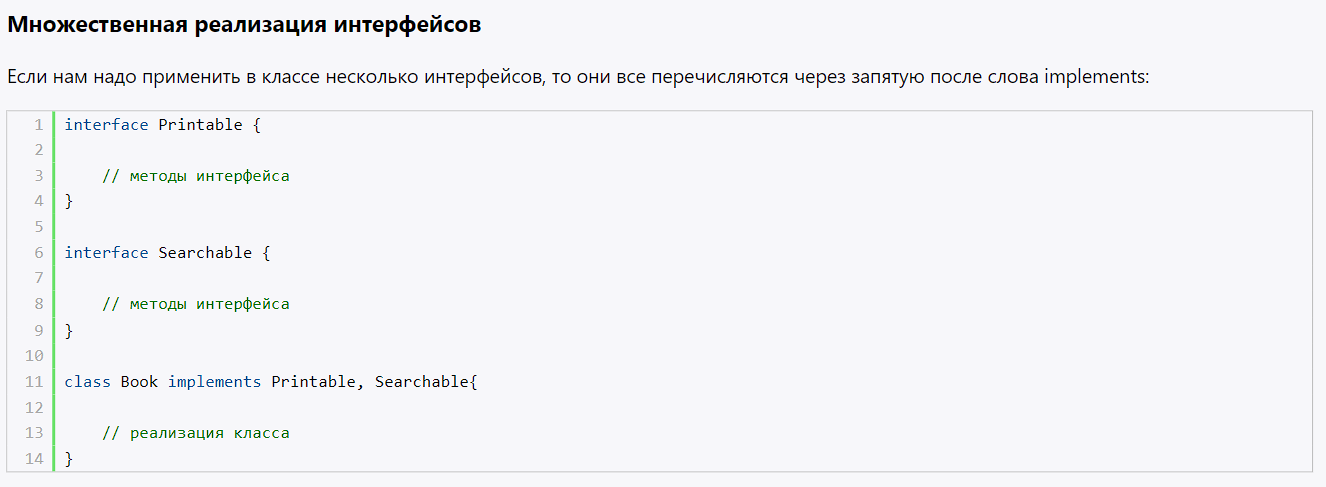
**50. Виды наследования в Джава, использование интерфейсов для реализации наследования**

****

****

****

Так как в Java отсутствует множественное наследование, данная механика реализуется инструментами интерфейсов. При наследовании класса от нескольких интерфейсов их стоит перечислить через запятую.



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

При наследовании новый класс получает доступ ко всем переменным и методам родительского класса, которые не помечены ключевым словом private.

Если класс помечен модификатором final, то от него наследоваться нельзя. Если же этим модификатором помечены поля и методы класса, то они могут быть использованы в дочернем классе, но не могут быть переопределены!

Также заметим, что при переопределении обязательно нужно вызвать конструктор родителя через super(), однако сам конструктор родительского класса переопределить нельзя!

**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**

* **public**: публичный, общедоступный класс или член класса. Поля и методы, объявленные с модификатором public, видны другим классам из текущего пакета и из внешних пакетов.
* **private**: закрытый класс или член класса, противоположность модификатору public. Закрытый класс или член класса доступен только из кода в том же классе.
* **protected**: такой класс или член класса доступен из любого места в текущем классе или пакете или в производных классах, даже если они находятся в других пакетах
* **Модификатор по умолчанию**. Отсутствие модификатора у поля или метода класса предполагает применение к нему модификатора по умолчанию. Такие поля или методы видны всем классам в текущем пакете.

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

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

String имя;

Как и в случае с типами int и double, переменные типа String можно инициализировать

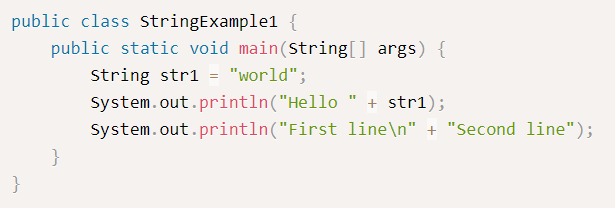
сразу при создании. Это, кстати, можно делать вообще со всеми типами в Java.

String name = "Аня", city = "New York", message = "Hello!";

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

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.

Использование:



**55. Что означает утверждение, что объект класса String является неизменяемым**

Методы могут только создавать и возвращать новые строки, в которых хранится результат операции.

**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**

Нет, StringBuilder не поддерживает конкатенацию строк, нужно использовать метод append().

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

Объявление массива - объявление его переменной.

Инициализация массива — заполнение массива конкретными значениями.

Есть два способа объявления массива:

1. **Первый способ** - объявление массива в Java-стиле:

String[] array; // объявление

1. **Второй способ** - объявление массива унаследованного стиля C/C++:

String array[]; // объявление

Также есть несколько способов инициализации массива:

1. **Первый способ -** инициализация массива после объявления:

array = new String[]{“hello”, ”world”, ”!”}; // инициализация

1. **Второй способ -** объявление и инициализация массива в одну строчку:

String array[] = new String[]{“hello”, ”world”, ”!”};

**ИЛИ**

String[] array = new String[]{“hello”, ”world”, ”!”};

**ИЛИ**

String[] array = {"abc", "def", "abcdef"};

**ИЛИ**

var s = new String[]{new String("Привет"), new String("Мир")};

Организация просмотра элементов массива - доступ к элементам массива по их индексам.

Есть три способа просмотра элементов массива:

1. Через цикл **while**:

int i = 0;

while (i < array.length){

System.out.println(array[i]);

i++;

}

1. Через цикл **for:**

for (int i = 0; i < array.length; i++)

System.out.println(array[i]);

1. Через цикл **foreach**:

for (var temp : array)

System.out.println(temp);

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

**Интерфейс в языке программирования Java** - это абстрактный тип, который используется для указания поведения, которое должны реализовывать классы. Невозможно создать объекты интерфейса, так как он абстрактный.

Чтобы определить интерфейс, используется ключевое слово **interface**:

interface Printable{

void print();

}

Все методы интерфейса не имеют модификаторов доступа, но фактически по умолчанию доступ public, так как цель интерфейса - определение функционала для реализации его классом.

Чтобы класс реализовал интерфейс, надо использовать ключевое слово **implements**:

class Book implements Printable{

String name;

String author;

Book(String name, String author){

this.name = name;

this.author = author;

}

public void print() {

System.out.printf("%s (%s) \n", name, author);

}

}

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

**Да**, в классе допускается реализовывать несколько интерфейсов, **если они не имеют одно и то же описание**. В этом случае все реализуемые в классе интерфейсы указываются списком через запятую. Пример:

interface Flyable {

fly();

}

interface Quackable {

quack();

}

class Duck implements Flyable, Quackable {

fly() { System.out.println(“Я лечу”); }

quack() { System.out.println(“Ква-ква”); }

}

**!** В противном случае реализовать интерфейсы в одном классе невозможно**. !**

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

Интерфейсы в Java являются ссылочными типами, как классы, но они могут содержать в себе только:

* константы
* сигнатуры методов
* default методы (методы по умолчанию)
* static методы (статические методы)
* вложенные типы.

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

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

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

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

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

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

Исключительная ситуация – это программная ошибка, которая возникает во время выполнения программного кода.

Обработка исключений происходит с помощью блока try-catch-finally:

try {

...

} catch (Exception e) {

...

}

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

Оператор throw следует использовать при возникновении ситуации, которую метод не может обработать самостоятельно.

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

Исключения делятся на:

* Наследники класса Error - критические ошибки, после которых программа не может восстановиться (например, переполнение программного стека);
* Наследники класса Exception - ошибки, которые можно обработать внутри приложения.

Error обрабатывает следующие ошибки: StackOverFlowError, OutOfMemoryError (JVM не может создать объект из-за нехватки памяти), AWTError (критическая ошибка AWT - оконной библиотеки JAVA).

Exception обрабатывает: EOFException (конец файла), FileNotFoundException, ArrithmeticException, MissingResourceException, ClassNotFoundException, NullPointerException, IllegalArgumentException, ArrayIndexOutOfBoundsException, UnknownTypeException.

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

Пример проблемного кода:

public class ArrayIndexOutOfBoundException {

public static void main(String[] args) {

String[] arr = { "A", "B", "C", "D" };

for(int i = 0; i < (arr.length + 3); i++) {

System.out.println(arr[i]);

}

}

}

java.lang.ArrayIndexOutOfBoundsException - доступ к массиву с недопустимым индексом.

**69. Контролируемые исключения (checked)**

Исключения бывают checked(проверяемые/контролируемые) и unchecked(непроверяемые/неконтролируемые).

Checked исключения, это те, которые должны обрабатываться блоком catch или описываться в сигнатуре метода (это имя метода плюс параметры, причем порядок параметров имеет значение, в сигнатуру метода не входит возвращаемое значение, а также бросаемые им исключения). Unchecked могут не обрабатываться и не быть описанными.

Checked исключения отличаются от Unchecked исключения в Java, тем что: наличие\обработка Checked исключения проверяются на этапе компиляции. Наличие\обработка Unchecked исключения происходит на этапе выполнения.

Проверяемые исключения обязательно нужно обрабатывать

либо пробрасывать. Непроверяемые — по желанию.

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

Исключения бывают checked(проверяемые/контролируемые) и unchecked(непроверяемые/неконтролируемые). Unchecked исключения в Java — наследованные от RuntimeException (исключения, которые наследуются от RuntimeException, — непроверяемые, а все остальные — проверяемые.). Unchecked исключения могут не обрабатываться и не быть описанными. Непроверяемые исключения (Unchecked) чаще всего являются следствием ошибок программиста.

| **Исключение** | **Описание** |
| --- | --- |
| ArithmeticException | Выбрасывается, когда возникает исключительная арифметическая ситуация. |
| ArrayIndexOutOfBoundsException | Выбрасывается при попытке обратиться к массиву по недействительному индексу. |
| ArrayStoreException | Выбрасывается при попытке сохранить объект несоответствующего типа в массиве объектов. |
| ClassCastException | Выбрасывается, когда код совершает попытку привести тип объекта к подклассу, экземпляром которого он не является. |
| ClassNotFoundException | Выбрасывается, когда приложение совершает попытку загрузить класс по его имени в строковом представлении с использованием метода forName в классе Class. |
| CloneNotSupportedException | Выбрасывается, когда для клонирования объекта определенного класса вызван метод clone, но класс этого объекта не реализует интерфейс Cloneable. |
| EnumConstantNotPresentException | Выбрасывается, когда приложение производит попытку обратиться к константе из перечисления по имени, но тип этого перечисления не содержит константу с указанным именем. |
| Exception | Класс Exception и его подклассы являются подклассами Throwable и указывают на ситуации, которые могут быть перехвачены приложением. |

Проверяемые исключения обязательно нужно обрабатывать либо пробрасывать. Непроверяемые — по желанию.

Checked исключения отличаются от Unchecked исключения в Java, тем что: наличие\обработка Checked исключения проверяются на этапе компиляции. Наличие\обработка Unchecked исключения происходит на этапе выполнения.

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

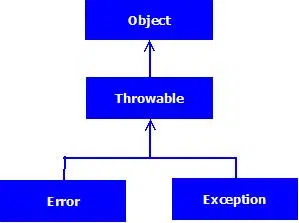
## **Иерархия исключений Java**

## **Исключения в Java делятся на две основные категории: встроенные и пользовательские.**

## Встроенные исключения

## **Встроенные исключения – это исключения, которые заранее определены в Java.**

## **Когда возникает встроенное исключение, виртуальная машина Java (JVM) создает объект, принадлежащий классу встроенного исключения. Все исключения происходят от класса java.lang.Throwable, но они определены в нескольких пакетах.**

****

**Класс Throwable происходит непосредственно от класса Object и является корневым классом дерева классов исключений. От него происходят два подкласса: Error и Exception. Ошибки и исключения, которые встречаются в программах на Java, являются объектами этих классов.**

**С использованием класса Throwable можно создавать собственные исключения.**

**Класс Error является надклассом для всех классов ошибок времени выполнения. Он завершает выполнение программы, если происходит ошибка, связанная с системой или ресурсами (JVM).**

**Ошибки обычно представляют собой необычную проблему, после которой сложно произвести восстановление. Они происходят не по вине программиста, а из-за ненадлежащей работы системы или выделения ресурсов.**

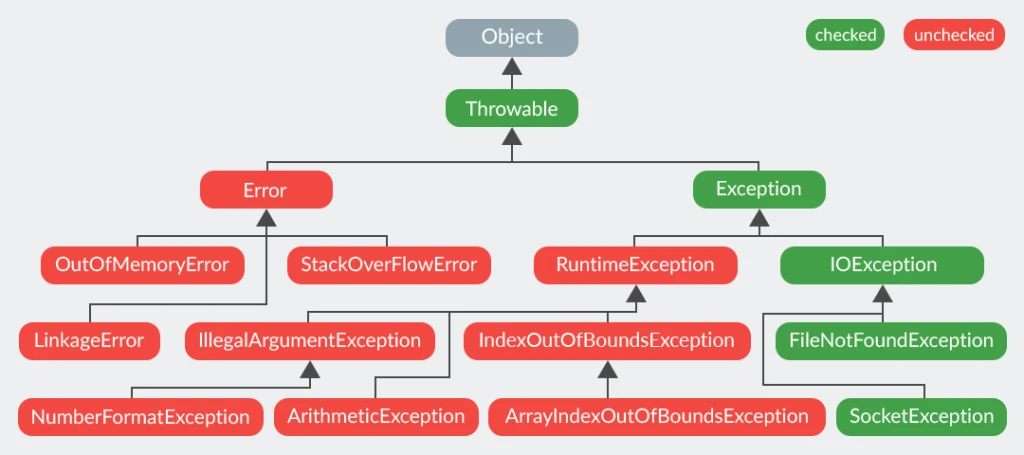
**Примеры ошибок: AssertionError, LinkageError, OutOfMmeoryError, StackOverFlowError, VirtualMachineError.**

**Класс Exception представляет ошибки, вызванные программой или внешними факторами. Это надкласс для все классов исключений.**

**Для этого класса существует два конструктора:**

* **public Exception() (по умолчанию)**
* **public Exception(String message) (принимает строковое сообщение как аргумент)**

**Эти конструкторы наследуются всеми подклассами исключений. Сам по себе класс Exception не предоставляет собственных методов. Он наследует методы класса Throwable.**

**Встроенные исключения делятся на две группы: проверяемые (checked) и непроверяемые (unchecked).**

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

Это оператор throw: throw new Exception();

**О чем говорит ключевое слово throws?**

Модификатор throws прописывается в сигнатуре метода и указывает на то, что метод потенциально может выбросить исключение с указанным типом.

Сигнатура метода — это **имя метода плюс параметры (причем порядок параметров имеет значение)**. В сигнатуру метода не входит возвращаемое значение, а также бросаемые им исключения.

****

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

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

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

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

**76. Объявление обобщённого 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**

ArrayList — реализация изменяемого массива интерфейса List, часть Collection Framework, который отвечает за список (или динамический массив), расположенный в пакете java.utils. Этот класс реализует все необязательные операции со списком и предоставляет методы управления размером массива, который используется для хранения списка.

В основе ArrayList лежит идея динамического массива. А именно, возможность добавлять и удалять элементы, при этом будет увеличиваться или уменьшаться по мере необходимости**.**

Он используется для хранения и изменения динамических наборов данных, таких как списки или массивы.

**78. Класс Pattern и его использование**

Класс Pattern представляет собой объект, который может быть использован для поиска и обработки текста. Он предоставляет методы для поиска и замены текста, а также для проверки соответствия строки шаблону. Это может быть полезно для автоматизации задач, таких как поиск и замена текста, проверка соответствия формату и т. д.

**79. Класс Math и его использование**

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 (значений по модулю)
* Вычисление значений тригонометрических функций (синусов, косинусов и т.д.)
* Возведение в различные степени
* Извлечение корней различных степеней
* Генерация случайных чисел
* Округления

Пример использования метода вычисления значения по модулю.

За это отвечает метод abs. Данный метод перегружен и в классе Math имеются следующие его различия:

* static double abs(double a)
* static float abs(float a)
* static int abs(int a)
* static long abs(long a)

Пример использования:

public static void main(String[] args) {

System.out.println(Math.abs(-1)); // 1

System.out.println(Math.abs(-21.8d)); // 21.8

System.out.println(Math.abs(4532L)); // 4532

System.out.println(Math.abs(5.341f)); // 5.341

}

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

Методы класса Math вызываются путем использования имени класса Math, а затем имени метода. Например, для вызова метода Math.abs() необходимо использовать Math.abs(). При вызове метода происходит выполнение определенной операции, которая зависит от метода. Например, метод Math.abs() возвращает абсолютное значение числа.

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

**81. Структура коллекций в Java Collection Framework. Иерархия интерфейсов**

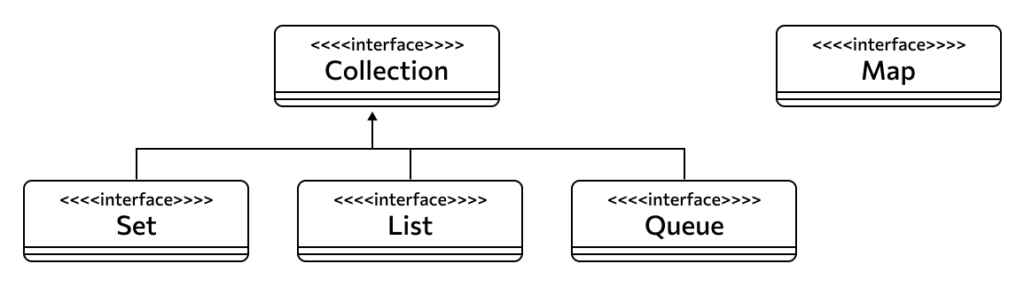
На вершине иерархии находится интерфейс Iterable (от него наследуется интерфейс Collection, от которого наследуется всё остальное, кроме Map (о Map позже))

Iterable объявляет метод Iterator<E> iterator(), который позволяет проходиться по коллекции циклом forEach. Возвращаемый объект типа Iterator имеет методы hasNext() и next(). С помощью них тоже можно итерировать по коллекции.

От интерфейса Collection<E> наследуются все остальные коллекции Java Collections Framework.

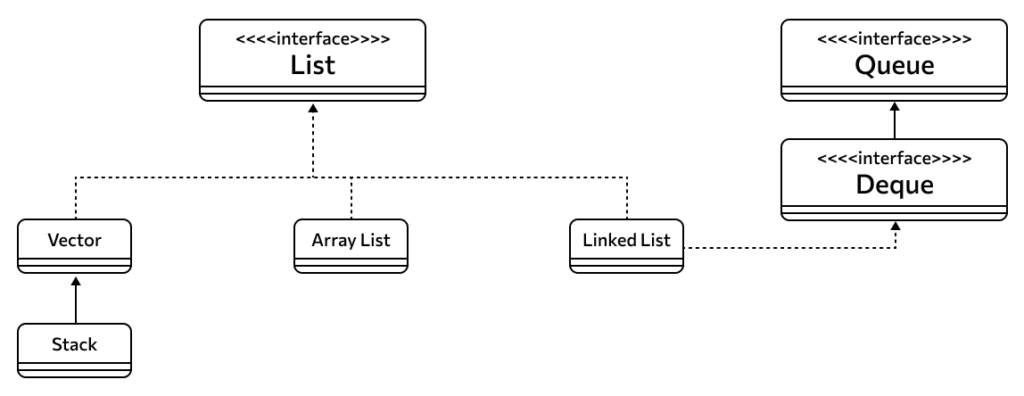
В этом интерфейсе объявлены методы:

* add(E)
* addAll(Collection<? extends E>)
* contains(E)
* containsAll(Collection<? extends E>)
* remove(E)
* retainAll(Collection<? extends E>) !!! логичнее было назвать removeAll но назвали retainAll
* clear()
* size()
* isEmpty()
* <T> T[] toArray(T[])



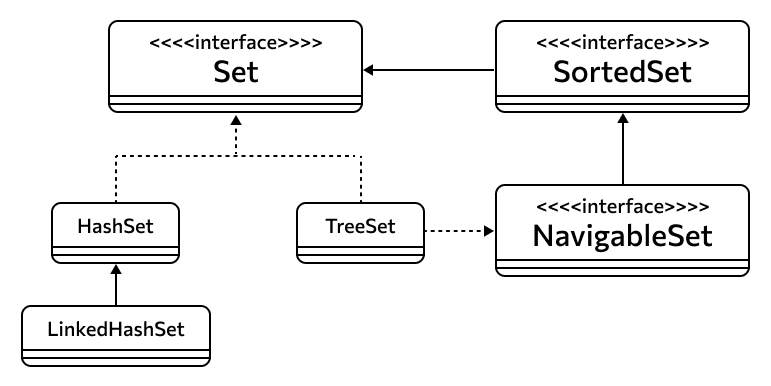
От интерфейса Collection наследуются следующие интерфейсы:

* List (список) — Упорядоченная коллекция (т. е. поддерживает индексацию элементов методом get(int))
* Set (набор) — Неупорядоченная коллекция (нет индексов), не может содержать двух и более одинаковых элементов.
* Queue (очередь) — Очередь)



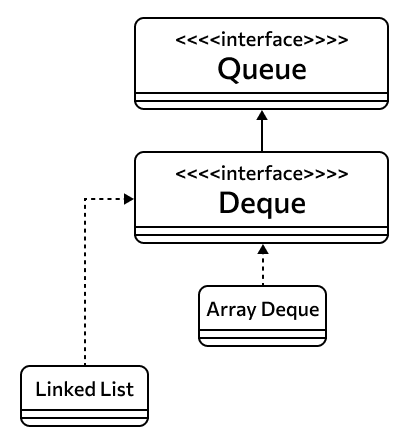
Классы реализующие интерфейс List:

* Vector — устарел. Используется, если нужна потокобезопасность. (Почти не отличается от ArrayList)
* ArrayList — основан на обычном массиве. При создании объекта класса ArrayList создаётся массив объектов размером 10. Когда места для добавления нового элемента нет, создаётся другой массив с большим объёмом, и все объекты из старого массива копируются в новый.
* LinkedList — двунаправленный связный список.
* Stack — стек)



Классы, реализующие интерфейс Set:

* HashSet — основан на HashMap. Объекты внутри неупорядочены (см. вопр. 87)
* LinkedHashSet — основан на LinkedHashMap. Объекты внутри упорядочены по времени добавления
* TreeSet — основан на красно-чёрном дереве

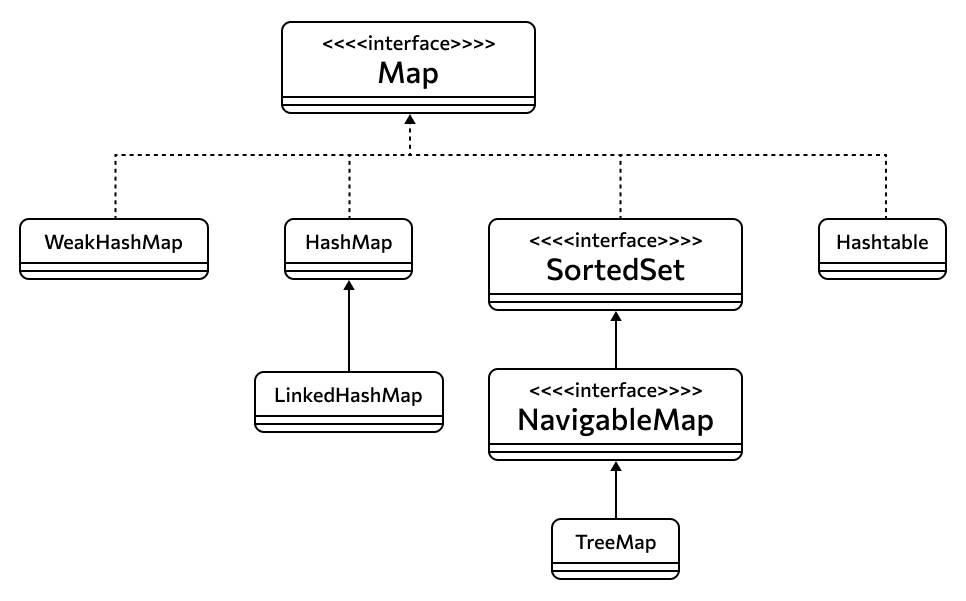


Классы, реализующие интерфейс Queue:

* LinkedList — двунаправленный связный список. [Реализует и интерфейс List и интерфейс Queue (Deque)].
* ArrayDeque — очередь на базе обычного массива.
* PriorityQueue — сортирует элементы в порядке добавления. Можно передать в конструктор объект типа Comparator, чтобы изменить это поведение.

Интерфейс Map

Map в иерархии интерфейсов находится отдельно от Iterable и Collection, но тоже является частью Java Collections Framework



Классы, реализующие интерфейс Map:

* Hashtable — устарела. Используется, если нужна потокобезопасность
* HashMap — замена Hashtable. Позволяет хранить null, неупорядочена. (см. вопр. 88)
* LinkedHashMap — как HashMap, но основан на двусвязном списке и позволяет итерировать по элементам в порядке их добавления (т. е. есть упорядоченность)
* TreeMap — основан на красно-чёрном дереве

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

Чтобы создать HashMap:

| HashMap<Integer, Integer> hashMap = new HashMap<>(); |
| --- |

Может содержать null как в качестве ключа так и в качестве значения

Метод put() добавляет элемент в HashMap:

| hashMap.put(1, 1);  hashMap.put(2, 4);  hashMap.put(3, 9); |
| --- |

Метод get() возвращает значение по ключу:

| hashMap.get(2); // => 4 |
| --- |

Метод containsKey():

| hashMap.containsKey(4); // => false |
| --- |

Метод containsValue():

| hashMap.containsValue(4); // => true |
| --- |

Метод remove() удаляет пару по ключу. Метод clear() очищает таблицу.

!!!ВАЖНО!!!

Чтобы проитерировать по HashMap:

| for(Map.Entry<Integer, String> ent : hashMap.entrySet()) {  System.out.println(ent.getKey() + "->" + ent.getValue());  } |
| --- |

Мы не можем итерировать HashMap непосредственно, для этого используется метод entrySet(), возвращающий набор пар из таблицы.

Каждая такая пара имеет тип Map.Entry<K, V>

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

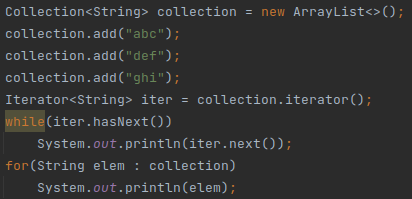
Коллекция на основе двунаправленного связного списка

Тут и добавить больше нечего ¯\\_(ツ)\_/¯

Внешне работает как любой другой List

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

Iterable (интерфейс, от которого наследуется Collection) объявляет метод Iterator<E> iterator(), который позволяет проходиться по коллекции циклом forEach. Возвращаемый объект типа Iterator имеет методы hasNext() и next(). С помощью них тоже можно итерировать по коллекции.



Цикл for можно написать так только потому что определён метод iterator()

**85. Что возвращает метод next()**

Метод E next() возвращает следующий (очередной) элемент коллекции.

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

Метод boolean hasNext() используется, чтобы проверять, есть ли еще элементы. Возвращает True или False

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

Обобщенный класс HashSet представляет хеш-таблицу. Он наследует свой функционал от класса AbstractSet, а также реализует интерфейс Set.

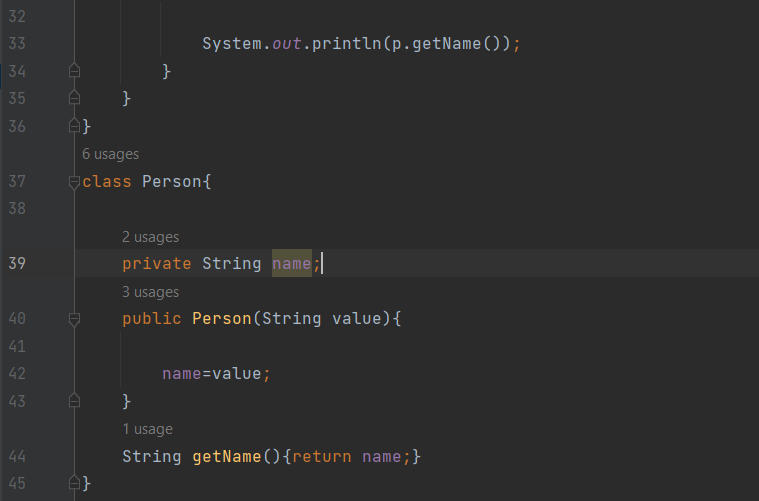
Хеш-таблица представляет такую структуру данных, в которой все объекты имеют уникальный ключ или хеш-код. Элементы хеш-таблицы хранятся в виде пар ключ-значение. Данный ключ позволяет уникально идентифицировать объект в таблице.

Для создания объекта HashSet можно воспользоваться одним из следующих конструкторов:

* HashSet(): создает пустой список
* HashSet(Collection<? extends E> col): создает хеш-таблицу, в которую добавляет все элементы коллекции col
* HashSet(int capacity): параметр capacity указывает начальную емкость таблицы, которая по умолчанию равна 16
* HashSet(int capacity, float koef): параметр koef или коэффициент заполнения, значение которого должно быть в пределах от 0.0 до 1.0, указывает, насколько должна быть заполнена емкость объектами прежде чем произойдет ее расширение. Например, коэффициент 0.75 указывает, что при заполнении емкости на 3/4 произойдет ее расширение.

Класс HashSet не добавляет новых методов, реализуя лишь те, что объявлены в родительских классах и применяемых интерфейсах:



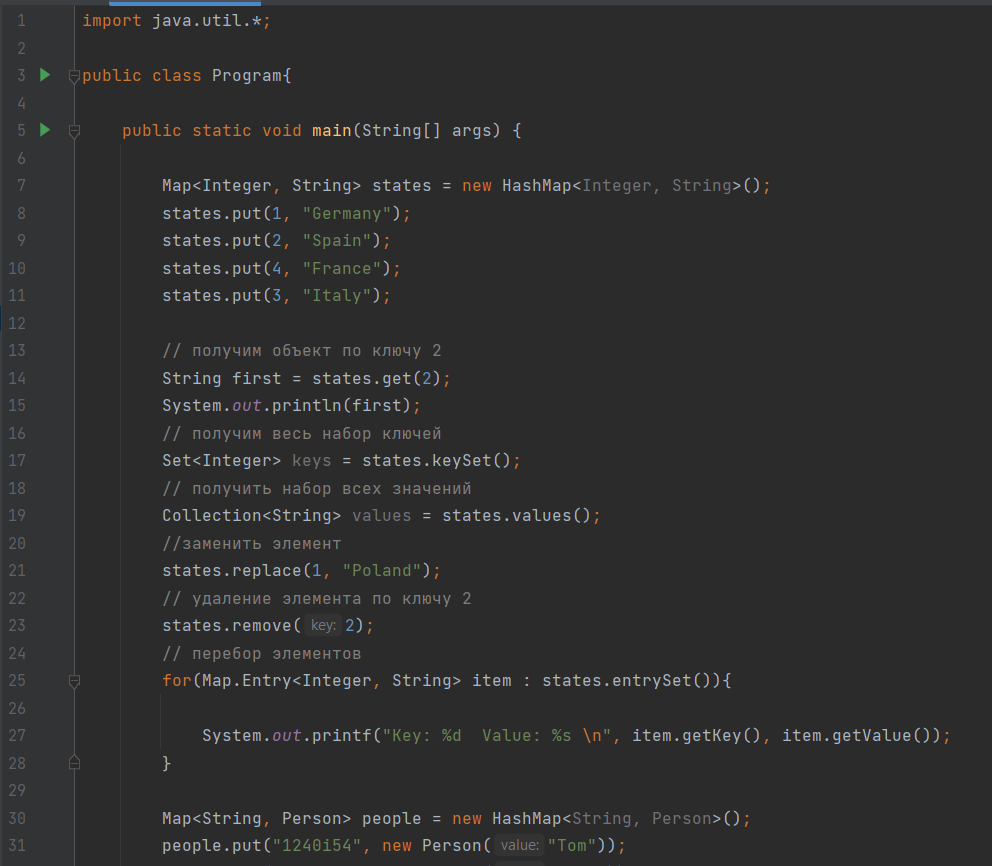


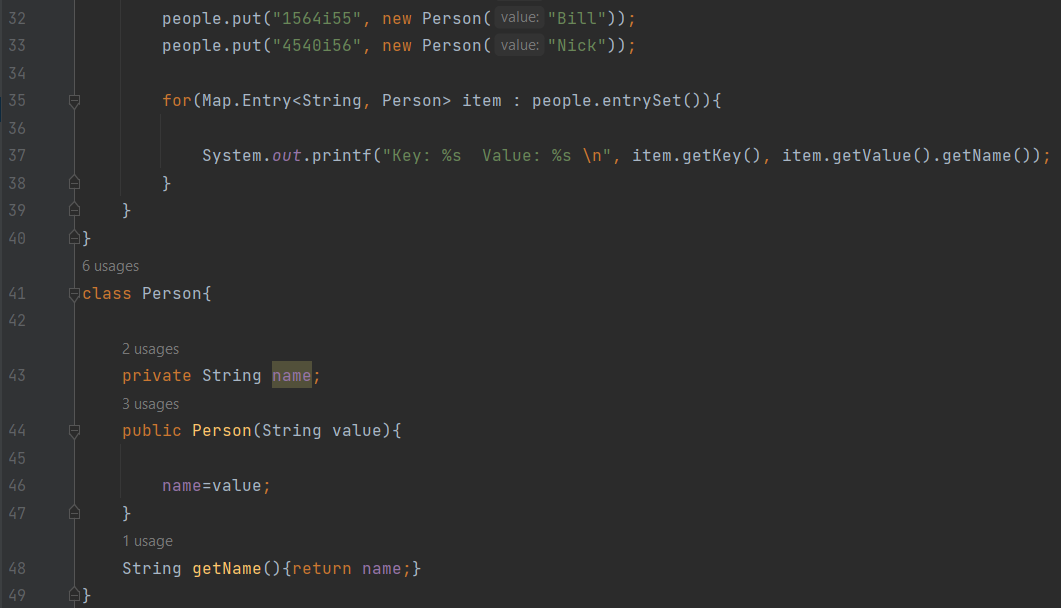
**88. Обобщенный класс HashMap класс коллекция, которая реализует интерфейс Map для хранения пар ключ-значение. Что он себя представляет?**

Класс HashMap реализует интерфейс Мар. Он использует хеш-таблицу для хранения карты. Это позволяет обеспечить константное время выполнения методов get() и put() даже при больших наборах. Ключи и значения могут быть любых типов, в том числе и null.

Ниже приведен список конструкторов, поддерживаемых классом HashMap:

* HashMap(): Этот конструктор создает стандартный HashMap по умолчанию
* HashMap(Map m): Этот конструктор инициализирует хэш-карту, используя элементы Map данного объекта m
* HashMap(int capacity): Этот конструктор инициализирует емкость хэш-карты для заданного целочисленного значения, capacity
* HashMap(int capacity, float fillRatio): Этот конструктор инициализирует как ёмкость, так и коэффициент заполнения хэш-карты, используя ее аргументы





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

**89. Стандартные поток ввода-вывода, предоставляемые Java**

Все классы, работающие с потоками байт, наследуются от абстрактных классов java.io.InputStream или java.io.OutputStream.

Класс Scanner(рассмотрен раньше)

Scanner scan = new Scanner (System.in);

answer = scan.nextLine();

hasNextInt(),nextInt()

System.out.print()

класс Console

Основные методы класса Console:

flush(): выводит на консоль все данные из буфера

format(): выводит на консоль строку с использованием форматирования

printf(): выводит на консоль строку с использованием форматирования (фактически то же самое, что и предыдущий метод)

String readLine(): считывает с консоли введенную пользователем строку

char[] readPassword(): считывает с консоли введенную пользователем строку, при этом символы строки не отображаются на консоли

import java.io.Console;

String name = System.console().readLine();

классы чтения с клавиатуры строк ввода BufferedReader и InputStreamReader, а также обрабатывает ошибки ввода с помощью класса обработки исключений IOException.

import java.io.BufferedReader; import java.io.IOException; import java.io.InputStreamReader;

BufferedReader br = new BufferedReader(new IputStreamReader(System.in)); System.out.print(“Enter String”);

String s = br.readLine(); System.out.print(“Enter Integer:”);

try { int i = Integer.parseInt(br.readLine()); } catch(NumberFormatException nfe) { System.err.println(“Invalid Format!”); } } }

java.io.DataInputStream и java.io.DataOutputStream

Классы DataInputStream и DataOutputStream позволяют платформонезависимо записывать в поток и считывать из потока примитивные типы языка Java. Класс DataInputStream реализует интерфейс DataInput, который содержит методы для чтения примитивных типов, аналогично класс DataOutputStream. Эти классы используют в качестве обёртки над другими потоками, например так:

**DataInputStream dis = new DataInputStream(new FileInputStream("myfile.data"));**

Методы интерфейса DataInput используют исключение java.io.EOFException для обозначения конца потока, в отличие от методов класса InputStream, которые возвращают -1.

java.io.ObjectInputStream и java.io.ObjectOutputStream

Классы ObjectInputStream и ObjectOutputStream позволяют считывать объекты из потока и записывать объекты в поток, то есть используются для [сериализации и десериализации объектов](https://urvanov.ru/2016/05/11/java-8-%d1%81%d0%b5%d1%80%d0%b8%d0%b0%d0%bb%d0%b8%d0%b7%d0%b0%d1%86%d0%b8%d1%8f/).

Сериализация:

import java.io.BufferedReader; import java.io.IOException; import java.io.InputStreamReader;

BufferedReader br = new BufferedReader( new IputStreamReader(System.in)); System.out.print(“Enter String”); String s = br.readLine(); System.out.print(“Enter Integer:”); try { int i = Integer.parseInt(br.readLine()); } catch(NumberFormatException nfe) { System.err.println(“Invalid Format!”); } } }

Десериализация:

try (ObjectInputStream ois = new ObjectInputStream (new FileInputStream("person.dat"))){

Person p=(Person)ois.readObject();

System.out.printf("Name: %s \t Age: %d \n", p.getName(), p.getAge());}

catch(Exception ex){System.out.println(ex.getMessage());}

**90. Понятие сериализации, интерфейс Serializable**

**Сериализация** представляет процесс записи состояния объекта в поток.

(процесс сохранения состояния объекта в последовательность байт)

**Десериализация** - процесс извлечения или восстановления состояния объекта из потока. Сериализация очень удобна, когда идет работа со сложными объектами.

**Serializable** - это маркерный интерфейс (не имеет элемента данных и метода). Он используется для “маркировки” классов java, чтобы объекты этих классов могли получить определенные возможности.

import java.io.Serializable;

class TestSerial implements Serializable {

public int c=0;}

Теперь объекты класса TestSerial можно сериализовать.

Сериализация:

public static void main(String args[]) throws IOException {

FileOutputStream fos = new FileOutputStream("temp.out");

ObjectOutputStream oos = new ObjectOutputStream(fos);

TestSerial ts = new TestSerial();

oos.writeObject(ts);

oos.flush();

oos.close();}

Десериализация:

Public static void main(String args[]) throws IOException {

FileInputStream fis = new FileInputStream("temp.out");

ObjectInputStream oin = new ObjectInputStream(fis);

TestSerial ts = (TestSerial) oin.readObject();

System.out.println("version="+ts.version);}

**91. Какие объекты можно сериализовать?**

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.

import java.io.Serializable;

class TestSerial implements **Serializable** {

public int c=0;}

Теперь объекты класса TestSerial можно сериализовать.

По умолчанию сериализуются все переменные объекта. Однако, возможно, мы хотим, чтобы некоторые поля были исключены из сериализации. Для этого они должны быть объявлены с модификатором transient.

**private transient double height;**

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

Интерфейс Serializable пуст.Для сериализации объектов в поток используется класс ObjectOutputStream. Для создания объекта ObjectOutputStream в конструктор передается поток, в который производится запись: ObjectOutputStream(OutputStream out). Для записи данных ObjectOutputStream использует ряд методов, среди которых можно выделить следующие:

void close(): закрывает поток

void flush(): очищает буфер и сбрасывает его содержимое в выходной поток

void write(byte[] buf): записывает в поток массив байтов

void writeObject(Object obj): записывает в поток отдельный объект

Класс ObjectInputStream отвечает за обратный процесс - чтение ранее сериализованных данных из потока. В конструкторе он принимает ссылку на поток ввода: ObjectInputStream(InputStream in). Основные методы этого класса:

void close(): закрывает поток

int skipBytes(int len): пропускает при чтении несколько байт, количество которых равно len

int available(): возвращает количество байт, доступных для чтения

int read(): считывает из потока один байт и возвращает его целочисленное представление

Object readObject(): считывает из потока объект

**93. Что означает понятие десериализация?**

**Десериализация** - процесс извлечения или восстановления состояния объекта из потока.

Десериализация:

Public static void main(String args[]) throws IOException {

FileInputStream fis = new FileInputStream("temp.out");

ObjectInputStream oin = new ObjectInputStream(fis);

TestSerial ts = (TestSerial) oin.readObject();

System.out.println("version="+ts.version);}

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

Его задачей является управление информацией о файлах и каталогах. Хотя на уровне операционной системы файлы и каталоги отличаются, но в Java они описываются одним классом File.

В зависимости от того, что должен представлять объект File - файл или каталог, мы можем использовать один из конструкторов для создания объекта:

File(String путь\_к\_каталогу)

File(String путь\_к\_каталогу, String имя\_файла)

File(File каталог, String имя\_файла)

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

Через конструктор класса FileOutputStream задается файл, в который производится запись. Класс поддерживает несколько конструкторов:

FileOutputStream(String filePath)

FileOutputStream(File fileObj)

*FileOutputStream fos=new FileOutputStream("C://SomeDir//notes.txt")*

*byte[] buffer = text.getBytes();*

*fos.write(buffer, 0, buffer.length);*

void write(byte[] buffer, int offset, int length): записывает в выходной поток некоторое число байтов, равное length, из массива buffer, начиная со смещения offset, то есть с элемента buffer[offset].

При записи строки, сначала переводим в массив байтов. С помощью метода write строка записывается в файл(в виде байтов).Используя перегрузку метода write(), можно записать и одиночный байт:

fos.write(buffer[0]); // запись первого байта