1. Объясните, что имеется в виду, когда говорится: Java-язык программирования и Java-платформа.

Java как платформа — это программное обеспечение, представляющее собой рабочую среду для работы программ, написанных на Java (и не только). Она состоит из Java API и Java виртуальной машины (JVM).

Java язык программирования является высокоуровневым, статически-типизированным и объектно-ориентированным.

2. Поясните, как связаны имя java-файла и классы, которые в этом файле объявляются.

Каждый класс хранится в отдельном файле. Имя файла совпадает с именем класса, расширение файла – java,  и в каждом файле написан код одного класса.

3. Расшифруйте аббревиатуры JVM, JDK и JRE; покажите, где “они находятся” и что собой представляют.

JVM- джава виртуальная машина, выполняет интерпретацию джава байт кода, специфицируется набором команд байт кода, набором регистров, стеком, сборщиком мусора и пространством хранения методов.

**JRE** – Java Runtime Environment, Java-окружение времени выполнения. Это набор всего необходимого чтобы запустить скомпилированную Java-программу. Окружение состоит из виртуальной машины Java (JVM), библиотеки классов (Java Class Library), консольной команды java и прочей инфраструктуры. Тем не менее, JRE не достаточно чтобы создавать новые программы.

**JDK** – Java Development Kit, набор Java-разработчика. Это полнофункциональный SDK для Java. JDK включает в себя JRE, но кроме того содержит компилятор (javac) и другие инструменты разработки, такие как javadoc и jdb. С помощью JDK можно создавать и компилировать программы.  
  
В большинстве случаев, если нам нужно только запускать Java программы, мы устанавливаем только JRE, если же мы планируем программировать на Java, вместо этого нам необходим JDK.

**4**. Объясните, как скомпилировать и запустить приложение из командной строки, а также зачем в переменных среды окружения прописывать пути к установленному jdk.

Открыть командную строку ввести java, пробел file name- название нашего файла, точка java, нажать enter.

Чтобы можно было из любого каталога, не имея необходимости вводить полный путь команды java файл на выполнение, иначе придется прописывать полный путь к файлу.

5. Перечислите атрибуты доступа, объясните их действие.

В Java используются следующие модификаторы доступа:

* **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.

6.Что такое пакеты в java-программе, что представляют собой пакеты на диске? Каково соглашение по именованию пакетов? Как создать пакет?

**Java** package (**пакет Java**) — механизм, позволяющий организовать **Java** классы в пространстве имен. **Java пакеты** могут содержаться в сжатом виде в JAR файлах. Обычно в **пакеты** объединяют классы одной и той же категории, либо предоставляющие сходную функциональность.

Пакеты на языке самого Java начинаются java. или javax.

Правой кнопкой мыши на папке **java** или на существующем пакете и выбираем в меню команду **New | Package**.

7.Объясните, какие классы, интерфейсы, перечисления необходимо импортировать в вашу программу, как это сделать. Влияет ли импорт пакета на импорт классов и др., лежащего в подпакетах? Какой пакет в Java импортируется по умолчанию?

Для использования класса в приложении, его следует подключить, при определении/создании нового объекта был указан пакет (полный путь к файлу). Однако данный подход не всегда удобен, и в качестве альтернативы можно импортировать пакеты и классы в приложение с помощью директивы **import**, которая указывается после директивы package:

Джава классы объединяются в пакеты. Пакеты позволяют организовать классы логически в наборы. По умолчанию java уже имеет ряд встроенных пакетов:

Java.lang- связывает фундаментальные классы

Java.io

Java.util- классы для ввода и выводные функции связаны в этом пакете

8.Объясните различия между терминами “объект” и “ссылка на объект”.

Объект - это экземпляр класса, а **ссылка** - это переменная, которая позволяет обратиться к объекту или классу, но в тоже время не является экземпляром класса.

9.Какие примитивные типы Java вы знаете, как создать переменные примитивных типов? Объясните процедуру, по которой переменные примитивных типов передаются в методы как параметры.

* boolean - булев тип, может иметь значения true или false
* byte - 8-разрядное целое число
* short - 16-разрядное целое число
* int - 32-разрядное целое число
* long - 64-разрядное целое число
* char - 16-разрядное без знаковое целое, представляющее собой символ UTF-16 (буквы и цифры)
* float - 32-разрядное число в формате IEEE 754 с плавающей точкой
* double - 64-разрядное число в формате IEEE 754 с плавающей точкой

Чтобы создать переменную указываются: **тип данных и имя переменной**.

Примитивные типы передаются по значению.

10.Каков размер примитивных типов, как размер примитивных типов зависит от разрядности платформы, что такое преобразование (приведение) типов и зачем оно необходимо? Какие примитивные типы не приводятся, ни к какому другому типу.

Byte -8 бит,short-16 бит, char-16 бит, int-32 бит, Long-64 бит, float -32 бит, double-64 бит,

Boolean-32 бит.

Спецификация Java абстрагируется от физического уровня и говорит лишь о том [какие значения могут принимать примитивы](https://javatalks.ru/out?url=https://docs.oracle.com/javase/specs/jls/se11/html/jls-4.html#jls-4.2.1) и какое байтовое представление будет видеть Java код - эти значения не зависят от разрядности. Но это также значит, что в зависимости от реализации JVM [кол-во байт которое физически отводится под тип может отличаться](https://javatalks.ru/out?url=https://shipilev.net/jvm/anatomy-quarks/23-compressed-references/).

В Java существует два типа преобразований - автоматическое преобразование (неявное)[расширяющее преобразование] и приведение типов (явное преобразование)[сужающее преобразование].

Она предназначено для удобного хранения данных. Данные из большого контейнера не могу храниться в меньшем.

Тип boolean **не приводится ни** к одному из **типов**.

11.Объясните, что такое явное и неявное приведение типов, приведите примеры, когда такое преобразование имеет место.

Для автоматического(неявного типа) преобразования типа должно выполняться два условия: оба типа должны быть совместимы, длина целевого типа должна быть больше длины исходного типа. В этом случае происходит преобразование с расширением.

byte -> short -> int -> long

int -> double

short -> float -> double

char -> int

При явном приведении типов сама операция приведения задается явным образом.

При автоматическом( неявном приведении) приведении типов нужно, чтобы выполнялись два условия:

* оба типа должны быть совместимыми;
* длина исходного типа (типа источника) должна быть меньше длины целевого типа (типа приемника).

Автоматическое продвижение типов происходит в выражениях. При этом значение, которое фигурируют в выражениях, автоматически продвигаются к типам с большими диапазонами значений.

При автоматическом продвижении типов в выражениях:

1- если один из целочисленных операндов имеет тип int, то все значения типов byte,

short и char продвигаются к типу int;

2 - если один из целочисленных операндов имеет тип long, то все выражение продвигается к

типу long;

3 - если один из операндов относится к типу float, то тип всего выражения будет также

типа float (если нет операндов типа double);

4 -если один из операндов относится к типу double, то тип всего выражения будет

также double.

12.Что такое литералы в Java-программе, какую классификацию литералов вы знаете, как записываются литералы различных видов и типов в Java-программе?

Литералы - явно заданные значения в коде программы - константы определенного типа, которые находятся в коде в момент запуска.

Все литералы — это примитивные значения (строки, числа, символы, булевы значения).

Нельзя создать литерал-объект. Единственный литерал, связанный с объектом — это null.

Литералы делятся на разделы:

1. Числовые:
   1. Целочисленные;
   2. С плавающей точкой;
2. Строковые;
3. Символьные;
4. Логические

**boolean** flag = false;

System.out.println("Амперсанд - " + '&');

**int** c = 456;

**float** d = 4.05E-13F

String str = ’’Это пример ’’;

Тип double в классическом виде: 2.718281828459045

Тип double в научном виде: 4.05E-13

13.Как осуществляется работа с типами при вычислении арифметически выражений в Java?

C помощью специальных символов, называемых **знаком операции**.

Есть унарные операции (выполняются над одним операндом), бинарные - над двумя операндами, а также тернарные - выполняются над тремя операндами. Операндом является переменная или значение (например, число), участвующее в операции.

Знаки операций, аргументами которых являются числа, разделяются на две категории: унарные (unary) знаки операций с одним аргументом и бинарные (binary) с двумя аргументами.

14.Что такое классы-оболочки, для чего они предназначены? Объясните, что значит: объект класса оболочки – константный объект.

 Классы-оболочки  предназначены не для вычислений, a для создания объектов, преобразования объектов, получения численных значений объектов в разных формах и передачи объектов в методы по ссылке. Объект класса оболочки нельзя изменить.

15.Объясните разницу между примитивными и ссылочными типами данных. Поясните существующие различия, при передаче параметров примитивных и ссылочных типов в методы. Объясните, как константные объекты ведут себя при передаче в метод.

Примитивные типы хранят значения.

Ссылочные переменные хранят адреса мест в памяти, где хранятся данные. Они предназначены для доступа к объектам.

Java передает все по значению, это значит - "скопировать значение и передать копию."

16.Поясните, что такое автоупаковка и автораспаковка.

**Автоупаковка** и **распаковка** это функция преобразования примитивных типов в объектные и наоборот. Весь процесс выполняется автоматически средой выполнения Java (JRE), следует быть осторожным при реализации этой функции, т.к. она может влиять на производительность программы.

**Автоупаковка** применяется компилятором Java в следующих условиях:

А).Когда значение примитивного типа передается в метод в качестве параметра метода,

который ожидает объект соответствующего класса-оболочки.

Б). Когда значение примитивного типа присваивается переменной, соответствующего класса оболочки.

Integer integer = 8;

**Распаковка** применяется компилятором Java в следующих условиях:

* Когда объект передается в качестве параметра методу, который ожидает соответствующий примитивный тип.
* Когда объект присваивается переменной соответствующего примитивного типа.

**int** in = 0;

in = **new** Integer(8);

17.Перечислите известные вам арифметические, логические и битовые операторы, определите случаи их употребления. Что такое приоритет оператора, как определить, в какой последовательности будут выполняться операции в выражении, если несколько из них имеют одинаковый приоритет.

Бинарные арифметические операторы(+,-,=,%,/).

Унарные арифметические оператор(+,-,++,--)

! — оператор отрицания;

&& — оператор логическое И (сокращенный);

|| — оператор логическое ИЛИ (сокращенный);

& — оператор побитовое И;

| — оператор побитовое ИЛИ;

^ — оператор побитовое исключающее ИЛИ.

Операция отрицания унарная и применяется к одному операнду. Все остальные операции — бинарные.

Логические операторы применимы только к boolean переменным.

Битовые операции производятся над битами. Двоичный код — набор нулей и единиц.

Побитовые операции в Java осуществляются только над целыми числами. А целые числа хранятся в памяти компьютера в виде набора битов. Компьютер переводит любую информацию в двоичную систему счисления (в набор битов) и только потом взаимодействует с ней.

& - and, >>, >>>, ^, ~ not, I - or, <<

Приоритет операторов определяет группирование терминов в выражении. Это влияет, как вычисляется выражение.

Существует таблица, в которой операторы с наивысшим приоритетом размещаются в верхней части, и уровень приоритета снижается к нижней части таблицы.

В выражении высокий приоритет операторов в Java будет оцениваться **слева направо.**

18.Укажите правила выполнения операций с плавающий точкой в Java (согласно стандарту IEEE754). Как определить, что результатом вычисления стала бесконечность или не число?

Числа с плавающей точкой (или действительные числа) представлены типами **float** и **double**. Используются для хранения значений с точностью до определенного знака после десятичной точки.

Стандарт IEEE 754 говорит нам, что представление действительных чисел должно записываться в экспоненциальном виде. Это значит, что часть битов кодирует собой так называемую мантиссу числа, другая часть — показатель порядка (степени), и ещё один бит используется для указания знака числа (0 — если число положительное, 1 — если число отрицательное). Математически это записывается так: **(-1)^s × M × 2^E**, где s — знак, M — мантисса, а E — экспонента.

**Стандарт IEEE 754 определяет:**

* как представлять нормализованные положительные и отрицательные числа с плавающей точкой
* как представлять денормализованные положительные и отрицательные числа с плавающей точкой
* как представлять нулевые числа
* как представлять специальную величину бесконечность (Infinity)
* как представлять специальную величину "Не число" (NaN или NaNs)
* четыре режима округления

**IEEE 754 определяет четыре формата представления чисел с плавающей запятой:**

* с одинарной точностью (single-precision) 32 бита
* с двойной точностью (double-precision) 64 бита
* с одинарной расширенной точностью (single-extended precision) >=43 бит (редко используемый)
* с двойной расширенной точностью (double-extended precision) >= 79 бит (обычно используют 80 бит)

**К операциям, приводящим к появлению NaN в качестве ответа, относятся:**

* все математические операции, содержащие NaN в качестве одного из операндов;
* деление нуля на нуль;
* деление бесконечности на бесконечность;
* умножение нуля на бесконечность;
* сложение бесконечности с бесконечностью противоположного знака;
* вычисление квадратного корня отрицательного числа[1];
* логарифмирование отрицательного числа.

**NaN** (Not-a-Number) данное состояние может возникнуть в различных случаях, например, когда предыдущая математическая операция завершилась с неопределённым результатом, или если в ячейку памяти попало не удовлетворяющее условиям число.

Infinity представляет собой математическую [бесконечность](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D1%81%D0%BA%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C#%D0%92_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B5) ∞. Это особое значение, которое больше любого числа.

Положительное число, разделенное на 0.0, дает «плюс бесконечность», а отрицательное – «минус бесконечность». Этим понятиям соответствуют специальные константы типа Double.

19.Что такое статический импорт, какие элементы можно импортировать при статическом импорте.

В Java имеется языковое средство, расширяющее возможности ключевого сло­ва import и называемое **статическим импортом**.

Оператор import, предваряемый ключевым словом static, можно применять для импорта статических членов класса или интерфейса. Благодаря статическому импорту появляется возможность ссылаться на статические члены непосредственно по именам, не уточняя их именем класса. Это упрощает и сокращает синтаксис, требующийся для работы со статическими членами. При объявлении статического импорта статические члены импортируются из классов, что позволяет им быть использованными без указания имени содержащего их класса.

20.Объясните работу операторов if, switch, while, do-while, for, for-each. Напишите корректные примеры работы этих операторов.

 1. Оператор **if** позволяет вашей программе в зависимости от условий выполнить оператор или группу операторов, основываясь на значении булевой переменной или выражения.

if (boolean\_значение) { /\* операторы \*/ } // 1

else { /\* операторы \*/ } // 2

Если выражение boolean\_значение принимает значение true, то выполняется группа операторов 1, иначе — группа операторов 2. Оператор else может отсутствовать, в этом случае операторы, расположенные после окончания оператора if (строка 2), вып

2. Цикл **fo**r следует использовать при необходимости выполнения алгоритма строго определенное количество раз.

3. **Оператор switch** — проверяет переменную на равенство в отношении списка значений. Каждое значение называется case, и переменная переключаясь проверяется для каждого case. Следующие правила применяются для оператора switch:

* Переменные, которые используются в операторе switch, могут быть только целые числа, конвертированные в целые числа (byte, short, char), строки и перечисления.
* Вы можете иметь любое количество операторов case в рамках одного switch. За каждым case следует сравниваемое значение, а затем идёт двоеточие.
* Значение case должен быть того же типа данных как и переменная в switch, и она должна быть константой или литерал.
* Когда переменная switch на равна оператору case, операторы следующие за case будут выполняться до тех пор, пока не будет достигнут оператор break.
* При достижении оператора break, switch завершается, и поток управления переходит к следующей строке после оператора switch.
* Не каждый case должен содержать break. Если отсутствует break, поток управления попадет на следующие case, до тех пор пока break не будет достигнут.
* В Java оператор switch может иметь дополнительный default case, который должен находиться в конце switch. Default case может быть использован для выполнения задачи, когда ни один из case является правильным. Break не требуется в default case

4. **Цикл do...while** — похож на  цикл while отличие, что цикл do...while гарантированно выполнится, по крайней мере один раз.  логическое выражение появляется в конце цикла, так что операторы в цикле выполнятся один раз, прежде чем пройдут проверку на логическое условие.

Если логическое выражение истинно, контроль переходит обратно, чтобы выполнить операторы, и они в цикле выполняются снова. Этот процесс повторяется до тех пор, пока логическое выражение не станет ложным

5. **while** — многократно выполняет цель оператора до тех пор, пока данное условие верно.

while(Логическое выражение)

{

//Операторы

}

Может быть один оператор или группа операторов. В условие может быть любое выражение, true(истинна) или любое не нулевое значение.

При выполнении, если результат логического выражения true, то действия внутри цикла будет выполняться. Это будет продолжаться до тех пор, пока результат выражения истинна.

Когда условие становится ложным, программа передаёт управление на строчку сразу после цикла.

6. For-each — это разновидность цикла for, которая используется, когда нужно обработать все элементы массива или коллекции.

**for** (type itVar : array)

{

Блок операторов;

}

Где type — тип итерационной переменной (совпадает с типом данных в массиве!), itVar — её имя, array — массив (тут также может быть другая структура данных, какая-нибудь коллекция, например, ArrayList), то есть объект, по которому выполняется цикл. Как вы видите, счётчик в такой конструкции не применяется, итерационная переменная перебирает элементы массива или коллекции, а не значения индекса.

При выполнении такого цикла, итерационной переменной последовательно присваивается значение каждого элемента массива или коллекции, после чего выполняется указанный блок операторов (или оператор).

For-each не подходит, если вы хотите внести изменения в массив. Например, не получится организовать сортировку массива, не меняя его элементы местами.

21. Объясните работу оператора instanceof. Что будет результатом работы оператора, если слева от него будет стоять ссылка, равная null?

**Оператор instanceof** — проверяет, является ли объект определенного типа (типа класса или типа интерфейса) и используется только для переменных ссылочного объекта. Оператор instanceof записывается в виде:

(Переменная ссылочного объекта) instanceof (класс/тип интерфейса).

Результатом действия оператора instanceof будет истина, если объект является объектом данного класса или одного из его подклассов, но не наоборот. Проверка на принадлежность объекта к классу Object всегда даст истину, поскольку любой класс является наследником класса Object. Результат применения этого оператора по отношению к ссылке на значение null — всегда ложь, потому что null нельзя причислить к какому-либо классу. В то же время литерал null можно передавать в методы по ссылке на любой объектный тип и использовать в качестве возвращаемого значения. Базовому типу значение null присвоить нельзя, так же как использовать ссылку на базовый тип в операторе instanceof.