Basics of software code development

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

Говоря о Java следует четко разграничивать два понятия: язык программирования и платформа.

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

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

JAVA-файл должен именоваться так же как и public class внутри него.

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

* JDK (Java Development Kit) - предоставляет среду для разработки и выполнения (запуска) Java-программы
* JRE (Java Runtime Environment) - это инсталляционный пакет, который предоставляет среду для запуска (но не разработки) java-программы (или приложения) на вашем компьютере. JRE используется только теми, кто хочет запускать только Java-программы, т.е. конечными пользователями вашей системы.
* JVM (Java Virtual Machine) является очень важной частью как JDK, так и JRE, потому что она содержится или встроена в обоих. Независимо от того, какую Java-программу вы запускаете с использованием JRE или JDK, она входит в JVM, а JVM отвечает за выполнение Java-программы построчно, поэтому она также называется интерпретатором

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

В командной строке выбрать директорию расположения файла программы. Затем набрать команду запустить JDK командой javac и имя файла программ с разрешением .java через пробел. После этого программа скомпилируется или выдаст ошибки. После удачного процесса компиляции появится файл с таким же названием, но с разрешением .сlass. Далее запускаем JRE командой java и имя файла без разрешения через пробел.

Переменные среды нужны для запуска программ через командную строку, в переменных среды прописывается путь к JDK

*5. Перечислите атрибуты доступа, объясните их действие*

private – наиболее строгий модификатор доступа, ограничивает видимость данных и методов предали одного класса, только в этом классе может быть доступ к эти данным.

default – модификатор доступа по умолчанию, он не прописывается перед переменными или методами, ограничивает доступ внутри пакета.

protected – переменные и методы, обозначенные этим модификатором, будут видны в переделах всех классов внутри пакета, а также в пределах всех классов-наследников.

public - переменные и методы с этим модификаторам видны во всех классах.

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

Пакет – это набор классов java , объединенных каким-либо логическим признакам, темой, свойством. Пакеты нужны для того, чтобы избежать конфликта имен классов. Чтобы указать, что класс принадлежит определенному пакету, надо использовать директиву package, после которой указывается имя пакета (package name\_ package). Названия пакетов соответствуют физической структуре проекта, то есть организации каталогов, в которых находятся файлы с исходным кодом. А путь к файлам внутри проекта соответствует названию пакета этих файлов. Пакет на диске – это папка с файлами-программами или еще с папкой-пакетом, внутри которого находятся также файлы-программы.

*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?*

Если надо использовать классы из других пакетов, то необходимо подключить эти пакеты и классы. Исключение составляют классы из пакета java.lang (например, String), которые подключаются в программу автоматически.

Чтобы импортировать пакет необходимо перед объявлением класса произвести импорт:

import PackageName.ClassFileName

PackageName – имя пакета

ClassFileName – имя файла с расширением .java, через точку могут перечисляться по иерархии подпакеты.

import PackageName.\* - импорт всего пакета PackageName.

После импорта пакета или класса нет необходимости в программе указывать полный путь к классу или объекту класса.

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

Объект – это сгруппированные вместе данные и методы для того, чтобы эти данные обрабатывать. У каждого объекта, как и у каждой переменной, есть тип. Этот тип определяется один раз при создании объекта и поменять его в дальнейшем нельзя. Типом объекта считается его класс.

Ссылка на объект – это указатель на этот объект. Ссылка не содержит объект, а только указывает на него.

Создание объекта: Class variable = new Class (attribute)

variable – это ссылка на объект. И для работы чаще используются ссылки на объект.

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

Различают 8 примитивных типов в Java:

- целочисленные типы:

byte – 8-разрядное целое число, от -128 до 127;

byte a;

a = -6;

byte b = 20;

short – 16-разрядное целое число, от -32768 до 32767;

short x;

short y=500;

int – 32-разрядное целое число, от -2147483648 до 2147483647;

int i, o;

int k = 50000;

long – 64-разрядное целое число, от -9223372036854775808 до 9223372036854775807;

long q= 300L;

long g;

- типы с плавающей точкой:

float - 32-разрядное число в формате IEEE 754 с плавающей точкой, от -3.4E+38 до 3.4E+38;

float f = 5.9F;

float k;

double - 64-разрядное число в формате IEEE 754 с плавающей точкой, от -1.7E+308 до 1.7E+308;

double x = 5.3E+2;

double z = 5.6;

- символы

char - 16-разрядное беззнаковое целое, представляющее собой символ, от 0 до 65536;

char a = 'a'

  char b = '\u1111'

- булевые типы

boolean – может иметь 2 значения – true и false.

boolean g = true;

boolean res = false;

Если методу передается аргумент примитивного типа, то происходит передача по значению. То есть, делается копия аргумента. Примеры:

static int Max(int x, int y){

if (x>y) return x;

else return y;

}

public static void Main (String[] args){

int x = 1;

int y = 2;

z = Max (x,y); // z = 2

}

или

public static void Main (String[] args){

z = Max (5,60); // z = 60

}

*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.*

Все размеры примитивных типов строго фиксированы и не зависят от машинной архитектуры. Это одна с причин улучшенной переносимости Java-программ.

byte – 8 бит;

short – 16 бит;

int – 32 бит;

long – 64 бит;

float – 32 бит;

double – 64 бит;

char  - 6 бит.

boolean – 4 байта;

В Java возможны преобразования между целыми значениями и значениями с плавающей точкой. Кроме того, можно преобразовывать значения целых типов и типов с плавающей точкой в значения типа char и наоборот, поскольку каждый символ соответствует цифре в кодировке Unicode. Тип boolean является единственным примитивным типом в Java, который нельзя преобразовать в другой примитивный тип. Кроме того, любой другой примитивный тип нельзя преобразовать в boolean.

Т.к. разные типы данных занимают разный объем памяти, то иногод необходимо преобразование одного типа данных в другой.

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

Преобразование типов в Java бывает двух видов: неявное и явное.

Неявное преобразование типов выполняется в случае если выполняются условия:

- Оба типа совместимы

- Длина целевого типа больше или равна длине исходного типа

Во всех остальных случаях должно использоваться явное преобразование типов.

Так же существуют два типа преобразований:

- Расширяющее преобразование

- Сужающее преобразование

Расширяющее преобразование происходит, если значение одного типа преобразовывается в более широкий тип, с большим диапазоном допустимых значений. Java выполняет расширяющие преобразования автоматически, например, если вы присвоили литерал типа **int** переменной типа **double** или значение переменной типа **char** переменной типа **int**. Неявное преобразование всегда имеет расширяющий тип.

int a = 2147483647;

float b = a;            // от типа int к типу float

Сужающее преобразование происходит, если значение преобразуется в значение типа, диапазон которого не шире изначального. Сужающие преобразования не всегда безопасны: например, преобразование целого значения 13 в **byte** имеет смысл, а преобразование 13000 в **byte** неразумно, поскольку **byte** может хранить только числа от −128 до 127. Поскольку во время сужающего преобразования могут быть потеряны данные, Java компилятор возражает против любого такого преобразования, даже если преобразуемое значение укладывается в более узкий диапазон указанного типа.

Сужающее преобразование это всегда явное преобразование типов.

int i = 13;  
byte b = (byte) i; // Принудительное преобразование int в byte

i = (int) 13.456; // Принудительное преобразование литерала типа double в int 13

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

Литералы — это 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. В соответствии с примитивами, литералы также делятся на разделы:

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

**Целочисленные литералы:**

Данный вид литералов — самый простой. Числа записываются в их стандартном виде без указывающих символов и прочего. Любое целое число — по умолчанию целочисленный литерал. То есть можно явно задать значение переменной или количество итераций в цикле. В Java поддерживается 4 системы счисления:

Двоичная

Восьмеричная

Десятичная

Шестнадцатеричная

Запись двоичного значения делается с помощью префикса 0b или 0B. Далее идет запись с использованием 0 и 1. Числа в восьмеричной системе записываются с помощью ведущего 0. Допустимые цифры — от 0 до 7. Запись 09 или 08 вызовет ошибку компиляции. С десятичной системой счисления нет никаких проблем: числа указываются в привычном нам виде. Единственное ограничение — число не может начинаться с 0, так как компилятор примет его за восьмеричное. Числа в шестнадцатеричной системе записываются с помощью префиксов 0x и 0X. Допустимые цифры — от 0 до 15, где числа 10-15 обозначаются символами A-F соответственно. Для записи больших чисел может использоваться символ нижнее подчеркивание «\_» для разделения по разрядам. По умолчанию тип литерала определяется как int.

int x=0B101; // вывод: число 5 (в двоичной системе)

int y=017; // вывод: число 15 (в восьмеричной системе)

int z=0X10f; //вывод: число 271 (в шестнадцатеричной системе)

int a=12; // вывод: число 12 (в десятеричной системе)

int b=1\_320\_508\_707; //вывод: число 1320508707

**Литералы с плавающей точкой:**

Числа с плавающей точкой, или дробные числа, могут записываться двумя способами.

Первый — в качестве классической десятичной дроби: 3.14159 или 2.718281282459045.

Второй — в научном виде, то есть обычная десятичная дробь плюс суффикс в виде символа е или Е и степени 10, на которую нужно умножить указанную дробь

По умолчанию тип литерала определяется как double. Для записи типа float в конце записи необходимо добавить f или F.

double x=3.14; // вывод 3.14

double y=2.3E-2; // вывод 0.023

float z=3.232124F; // вывод 3.232124

float a=2.65E+3F; // вывод 2650.0

**Строковые литералы**

Строковые литералы — это набор символов, заключенных в двойные кавычки. Данный тип используется так же часто, как и числовые литералы. В строке также могут находится служебные символы, которые необходимо экранировать (так называемые escape-последовательности).

String st = “Hello, Java”; // вывод Hello, Java

**Символьные литералы**

Символьные литералы в Java представлены кодовой таблицей Unicode, то есть каждый символ — 16-битовое значение. Для обозначения символа в коде его обособляют одинарными кавычками. Бывают два вида символов:

- те, что можно ввести с клавиатуры — обычные символы;

- символы, которые просто так с клавиатуры не ввести (символы различных языков, фигуры и так далее).

Обычные символы можно указать в явном виде: ‘,’ или ‘@’. Если символ служебный (например, перенос строки или табуляция), такой символ необходимо экранировать обратным слешем. Символы, которые нельзя просто так ввести с консоли, можно задать в их 16-битовом виде. Для этого необходимо указать код символа с префиксом '\u'. Также символы можно указывать в восьмеричном стиле (трехзначный номер), добавив в начале просто обратный слеш.

char x='%'; // вывод %

char y='\u00f8'; // вывод Ø

char z='\101'; // вывод А

**Самый простой литерал — это логический.** Имеется всего 2 значения: false и true, которые указываются явно без различных символов. Такие литералы могут быть присвоены переменным типа boolean или указаны в месте, где ожидается тип boolean.

boolean a = true;

boolean b = false;

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

В Java определены следующие арифметические бинарные операции:

сложение "+";

вычитание "-";

умножение "\*";

деление "/";

вычисление остатка от деления целых чисел "%" (возвращает остаток от деления первого числа на второе, причем результат будет иметь тот же знак, что и делимое), например, результат операции 5%3 будет равен 2, а результат операции (-7)%(-4) будет равен -3. В Java операция может использоваться и для вещественных переменных (типа float или double).

Унарные операции

* инкремент "++" (только для целых) – увеличивает значение переменной на 1;
* декремент "--" (только для целых) – уменьшает значение переменной на 1.
* Когда операция деления выполняется над целочисленным типом данных, ее результат не будет содержать дробный компонент, дробная часть будет отброшена. Результат арифметической операции над целыми числавыми имеет тип int, кроме того случая, когда один из операндов типа long. В этом случае результат будет типа long. Операции над числами с плавающим нулем по умолчанию double.

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

Java - объектно-ориентированный язык программирования. Это значит, что в ней "все, что можно, представлено в виде объектов". Поэтому, у примитивных типов есть объекты-аналоги - так называемые "классы оболочки". Обычные примитивы занимают меньше места в памяти и поэтому над ними можно проводить операции быстрее. Но над объектами аналогичных классов-оболочек можно производить большее количество операций и специальных действий, например, перевести текст в число (.parseInt). Эти операции, методы уже прописаны в классе-оболочке.

Константный объект - это значение примитива, для которого объект является оболочкой. К примеру, для int - оболочка java.lang.Integer для long - оболочка - java.lang.Long. В исходном коде класса java.lang.Integer имеется приватная финальная переменная типа int -  private final int value, которую нельзя изменить. То есть объявляя Integer x = 1; на самом деле, выполняется нечто вроде:

Integer x = new Integer(1)

Далее, если мы захотим поменять значение x = 2, то переменная x будет указывать уже на другой объект, а не перезатрет значение предыдущего. Предыдущий объект, будет собран сборщиком мусора, если на него более не имеется ссылок.

*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.*

Каждая [переменная в Java](https://javahelp.online/osnovy/peremennye-java) имеет тип данных, которые разделены на две группы: примитивные и ссылочные.

Примитив содержит значение переменной непосредственно в памяти, выделенной для нее.

Ссылка не содержит объект, но содержит ссылку на него, которая указывает на другое место в памяти, где он хранится. Через такую ссылку можно получить доступ к полям и методам ссылочного объекта. Допустимо иметь много разных переменных, ссылающихся на один и тот же объект. Это невозможно сделать с примитивами.

В Java существует 2 способа для передачи переменной или объекта класса в функцию:

- *передача по значению.* В этом случае значение аргумента копируется в формальный параметр функции. Поскольку создается копия аргумента в функции, то все изменения над копией не повлияют на значение аргумента;

- *передача по ссылке (по адресу).* В этом случае параметру передается ссылка на аргумент, который используется при вызове. По этой ссылке есть доступ к аргументу. Таким образом, все изменения, сделанные в теле функции над значением параметра, будут изменять значение аргумента, который был передан в функцию.

Если методу передается аргумент примитивного типа, то происходит передача по значению. То есть, делается копия аргумента.

В отличие от переменных примитивных типов, объекты класса передаются по ссылке. Это значит, что изменения, сделанные в теле функции (методе) будут изменять также значения объекта, который задавался в качестве аргумента.

Кроме переменных, в Java для хранения данных можно использовать константы. В отличие от переменных константам можно присвоить значение только один раз. Константа объявляется так же, как и переменная, только вначале идет ключевое слово final. Константы позволяют задать такие переменные, которые не должны больше изменяться.

*16. Поясните, что такое автоупаковка и автораспаковка.*

Преобразования примитивных типов в эквивалентные объекты в Java реализуется автоматически. Это свойство известно как **Автоупаковка** (Autoboxing). Обратный процесс соответственно – **Распаковка** (Unboxing) т.е. процесс преобразования объектов в соответствующие им примитивные типы.

Автоупаковка: Integer integer = 9;

Автораспоковка: int in = 0;

in = new Integer(9);

Автоупаковка применяется компилятором Java в следующих условиях:

* Когда значение примитивного типа передается в метод в качестве параметра метода, который ожидает объект соответствующего класса-оболочки.
* Когда значение примитивного типа присваивается переменной, соответствующего класса оболочки.

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.

*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.*

Операторы в языке Java — это специальные символы, которые сообщают транслятору о том, что вы хотите выполнить операцию с не­которыми операндами. Некоторые операторы требуют одного операн­да, их называют унарными. Одни операторы ставятся перед операндами и называются пре­фиксными, другие — после, их называют постфиксными операторами. Большинство же операторов ставят между двумя операндами, такие операторы называют­ся инфиксными бинарными операторами. Существует тернарный опе­ратор, работающий с тремя операндами.

 В Java имеется 44 встроенных оператора. Их можно разбить на 4 класса - арифметические, битовые, операторы сравнения  и логические.

**Арифметические операторы.**

Используются для вычислений так же как в алгебре. Допустимые операнды должны иметь числовые типы. Например, исполь­зовать эти операторы для работы с логическими типами нельзя, а для работы с типом char можно, поскольку в Java тип char — это подмно­жество типа int.

+ - оператор сложение: int a = 5 + 3; // результат 8

- - оператор вычитания (унарный минус): int a = 5 – 3;// результат 2

\* - умножение: int a = 5 \* 3; // результат 15

/ - оператор деления: int a = 5 / 3; // результат 1

% - деление по модулю: int a = 5% 3; результат 2

++ - инкремент (прибавляет 1):

int b = 5;

int a=++b; // результат 6

int c=b++; // результат 5

-- - декремент (отнимает 1):

int b = 5;

int a=--b; // результат 4

int c=b--; // результат 5

+= - сложение с присваиванием:

int a = 5;

a += 5; // результат 10

-= - вычитание с присваиванием:

int a = 5;

a -= 5; // результат 0

\*= - умножение с присваиванием:

int a = 5;

a \*= 5; // результат 25

/= - деление с присваиванием:

int a = 5;

a /= 5; // результат 1

%= - деление по модулю с присваиванием:

int a = 5;

a += 5; // результат 0

= - оператор присваивания

int a = 5; // результат 5

### Целочисленные битовые операторы

Для целых числовых типов данных — long, int, short, char и byte, определен дополнительный набор операторов, с помощью которых можно проверять и модифицировать состояние отдельных битов соответствую­щих значений. Операторы битовой арифметики работают с каждым битом как с самостоятельной величиной.

⁓ - побитовое унарное отрицание (NOT): int x = ⁓ 5; // результат -6

& - побитовое И (AND): int x = 6&5; // результат 4

| - побитовое ИЛИ (OR): int x = 6|5; // результат 7

^ - побитовое исключающее ИЛИ (XOR): int x = 6|5; // результат 3

&= - побитовое И (AND) с присваиванием:

int x = 5;

x &= 6; // результат 4

| = - побитовое ИЛИ (OR) с присваиванием:

int x = 5;

x |= 6; // результат 7

Ис­пользование операторов в программе на языке Java.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| А | В | OR | AND | XOR | NOT A |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

^= - побитовое исключающее ИЛИ (XOR) с присваиванием:

int x = 5;

x ^= 6; // результат 3

>> - сдвиг вправо: byte x = (byte) (100 >> 5); // результат 3

<< - сдвиг влево: byte x = (byte) (100 << 5); // результат -128

>>> - сдвиг вправо с заполнением нулями: byte x = (byte) (50<<5);//результат 1

>>= - сдвиг вправо с присваиванием:

byte a = 100;

a >>=5; // результат 3

<<= - сдвиг влево с присваиванием:

byte a = 100;

a <<=5; // результат -128

>>>= - сдвиг вправо с заполнением нулями с присваиванием:

byte a = 50;

a >>>=1; // результат 25

**Операторы отношения**

Для того, чтобы можно было сравнивать два значения, в Java имеется набор операторов, описывающих отношение и равенство.

== - равно: boolean a = 5 == 6; // результат false

!= - не равно: boolean a = 5 != 6; // результат true

> - больше: boolean a = 5 > 6; // результат false

< - меньше: boolean a = 5 < 6; // результат true

>= - больше или равно: boolean a = 5 >= 6; // результат false

<= - меньше или равно: boolean a = 5 >= 5; // результат true

**Булевы логические операторы**

Оперируют только с операндами типа boolean. Все бинарные ло­гические операторы воспринимают в качестве операндов два значения типа boolean и возвращают результат того же типа.

& - логическое И (AND):

boolean a = true;

boolean b = false;

boolean c = a & b; //результат false

| - логическое ИЛИ (OR):

boolean a = true;

boolean b = false;

boolean c = a | b; //результат true

^ - логическое исключающее ИЛИ (XOR):

boolean a = true;

boolean b = false;

boolean c = a ^ b; //результат true

|| - оператор OR быстрой оценки выражений:

| - логическое ИЛИ (OR):

boolean a = true;

boolean b = false;

boolean c = a || b; //результат true

&& - оператор AND быстрой оценки выражений:

| - логическое ИЛИ (OR):

boolean a = true;

boolean b = false;

boolean c = a && b; //результат false

! – логическое унарное отрицание (NOT):

boolean a = true;

boolean c = !a; //результат false

&= - И (AND) с присваиванием:

boolean a = true;

boolean b = false;

a &= b; //результат false

|= - ИЛИ (OR) с присваиванием:

boolean a = true;

boolean b = false;

a |= b; //результат true

^= - исключающее ИЛИ (XOR) с присваиванием:

boolean a = true;

boolean b = false;

a ^= b; //результат true

?: - тернарный оператор if-then-else:

int a = 5;

int b = 6;

int c = (a>b)?a\*b:a+b;//результат 11

если a больше b, то c равно a умножить на b, а противном случае c равно a плюс b.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| А | В | OR | AND | XOR | NOT A |
| false | false | false | false | false | true |
| true | false | true | false | true | false |
| false | true | true | false | true | true |
| true | true | true | true | false | false |

В таблице ука­заны в порядке убывания приоритеты всех операций языка Java.

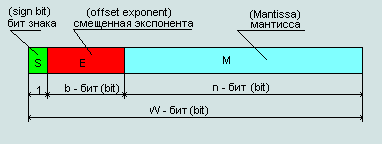
|  |  |  |  |
| --- | --- | --- | --- |
| Высший | | | |
| ( ) | [ ] | . |  |
| ~ | ! |  |  |
| \* | / | % |  |
| + | - |  |  |
| >> | >>> | << |  |
| > | >= | < | <= |
| == | != |  |  |
| & |  |  |  |
| ^ |  |  |  |
| | |  |  |  |
| && |  |  |  |
| | | |  |  |  |
| ?: |  |  |  |
| = | op= |  |  |
| Низший | | | |

Если используются операторы одинакового уровня, то приоритет будет у того оператора, который находится левее.

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

 В наиболее распространённом формате (стандарт IEEE 754) число с плавающей запятой представляется в виде набора битов, часть из которых кодирует собой мантиссу числа, другая часть — показатель степени, и ещё один бит используется для указания знака числа (0 — если число положительное, 1 — если число отрицательное). При этом порядок записывается как целое число в [коде со сдвигом](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%86%D0%B5%D0%BB%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB:_%D0%BF%D1%80%D1%8F%D0%BC%D0%BE%D0%B9_%D0%BA%D0%BE%D0%B4,_%D0%BA%D0%BE%D0%B4_%D1%81%D0%BE_%D1%81%D0%B4%D0%B2%D0%B8%D0%B3%D0%BE%D0%BC,_%D0%B4%D0%BE%D0%BF%D0%BE%D0%BB%D0%BD%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), а мантисса — в [нормализованном виде](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B2%D0%B5%D1%89%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB#.D0.9D.D0.BE.D1.80.D0.BC.D0.B0.D0.BB.D1.8C.D0.BD.D0.B0.D1.8F_.D0.B8_.D0.BD.D0.BE.D1.80.D0.BC.D0.B0.D0.BB.D0.B8.D0.B7.D0.BE.D0.B2.D0.B0.D0.BD.D0.BD.D0.B0.D1.8F_.D1.84.D0.BE.D1.80.D0.BC.D0.B0), своей дробной частью в двоичной системе счисления.

Формальное представление чисел в стандарте IEEE 754 для любого формата точности.



где:

S - бит знака, если S=0 - положительное число; S=1 - отрицательное число

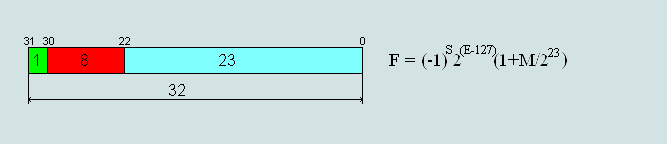
E - смещенная экспонента двоичного числа; exp2 = E - (2(b-1) - 1) - экспонента двоичного нормализованного числа с плавающей точкой (2(b-1) -1) - заданное смещение экспоненты (в 32-битном IEEE754 оно равно +127)

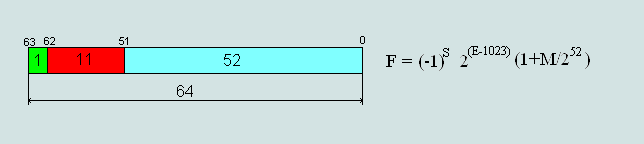
M - остаток мантиссы двоичного нормализованного числа с плавающей точкой

Формула вычисления десятичных чисел с плавающей точкой, из чисел представленных в стандарте IEEE754:

Формула нормализованых чисел IEEE754 

Используя формулу можно вычислить формулы для нахождения десятичных чисел из форматов одинарной (32 бита) и двойной (64 бита) точности IEEE 754:

Формат числа одинарной точности (single-precision) 32 бита

Формат числа двойной точности (double-precision) 64 бита

В число с плавающей запятой можно записать значение +∞ или −∞. Как и нули со знаком, бесконечности позволяют получить хотя бы близкий к правильному результат вычисления в случае переполнения. Согласно стандарту IEEE 754 число с плавающей запятой считается равным бесконечности, если все двоичные разряды его порядка — единицы, а мантисса равна нулю. Знак бесконечности определяется знаковым битом числа.

Нечисло NaN ("not a number") является результатом арифметических операций, если во время их выполнения произошла ошибка. В IEEE 754 NaN представлен как число, в котором все двоичные разряды порядка — единицы, а мантисса не нулевая. Любая операция с NaN возвращает NaN. При желании в мантиссу можно записывать информацию, которую программа сможет интерпретировать. Стандартом это не оговорено и мантисса чаще всего игнорируется.

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

Для статического импорта используется оператор Java import static, после которого указывается полное имя класса и метод или переменная. 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.

Пример без использований static import:

double value = Math.cos(Math.PI\*4);

После применение static import:

double value = cos(PI\*4);

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

**Оператор *if-else*:**

int x = 1;

int y = 2;

int result;

if (x > 0){ // внутри скобок указывается условие

result = x + y; // если условие true – выполняется данное действие

}

else if (x == 0){ // внутри скобок указывается еще одно условие

result = 0; // если новое условие true – выполняется данное действие

} else {

result = y - x; // если условие false – выполняется данное условие

}

**Оператор *switch*:**

int numberDay = 1;

String dayOfWeek;

switch (numberDay){ //выражение, переменная для сравнения с case (как правило целое число)

case 1: dayOfWeek = “Понедельник”; //проверяется совпадение с 1, если совпадает, то выполняется команда, если нет, то сравнение продолжается

break; //при условии совпадения происходит выход из оператора switch

case 2: dayOfWeek = “Вторник”; // по аналогии с case 1

break;

case 3: dayOfWeek = “Среда”;

break;

case 4: dayOfWeek = “Четверг”;

break;

case 5: dayOfWeek = “Пятница”;

break;

case 6: dayOfWeek = “Суббота”;

break;

case 7: dayOfWeek = “Воскресенье”;

break;

default: dayOfWeek = “День не определен”; // если не найдено совпадений,

то выполняется команда

после default

break;

}

**Оператор *while*:**

int x = 0;

int y = 1;

int result;

while (x < 5){ //циклический оператор while, внутри скобок указывается условие, если оно true, то выполняется тело цикла до тех пор, пока условие не будет false

result = x + y; //команда для выполнения при true

x++;} // команда для выполнения при true, увеличиваем х на 1.

**Оператор *do-while*:**

do-while отличается от while только тем, что тело цикла выполнится хотя бы один раз, т.е. сначала выполняется тело цикла, а потом идет сравнение.

int x = 0;

int y = 1;

int result;

do {

result = x + y; //команда для выполнения при true в условии while

x++;} // команда для выполнения при true, увеличиваем х на 1.

while (x < 5); //циклический оператор while, внутри скобок указывается условие, если оно true, то выполняется тело цикла до тех пор, пока условие не будет false

**Оператор *for*:**

int sum = 0;

for (int i = 0; i < 10; i++){ //циклический оператор for, внутри скобок указывается: инициализируется переменная и ее начальное значение; условие выполнения цикла; действие над переменной после выполнения тела цикла)

sum = sum + i; } //команда, которая будет выполнятся пока в условие выполнения true

**Оператор *for-each:***

for-each – это оператор для перебора массива/коллекций, разновидность оператора for

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

int sum = 0;

for (int i:array) { //циклический оператор for, внутри скобок указывается: тип и название (int i) : название массива/коллекции (array))

sum = sum + i; } //команда для исполнения.

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

Оператор instanceof сравнивает объект с указанным типом. Можно использовать его для проверки, является ли объект экземпляром класса, экземпляром подкласса или экземпляром класса, который реализует определенный интерфейс. Возвращает значение boolean.

String s = “Hello”;

return s instanceof String; //result = true

Применение instanceof в нулевой ссылочной переменной возвращает false.

String s = null;

return s instanceof String; //result = false