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

Ответ:

Java (язык) – высокоуровневый [строго](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%B8_%D1%81%D0%BB%D0%B0%D0%B1%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [типизированный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [объектно-ориентированный язык программирования](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения. Основная особенность - язык транслируется в байт-коде, который интерпретируется JVM. Тем самым реализуется принцип - написано один раз, работает везде.

Платформа [Java](https://ru.bmstu.wiki/Java) - это набор программ, которые облегчают разработку и запуск программ, написанных на языке программирования [Java](https://ru.bmstu.wiki/Java). Платформа [Java](https://ru.bmstu.wiki/Java) будет включать механизм выполнения (JVM), компилятор и набор библиотек. [Java](https://ru.bmstu.wiki/Java) не относится ни к одному процессору или операционной системе, поскольку платформы [Java](https://ru.bmstu.wiki/Java) были реализованы для широкого спектра аппаратных и операционных систем, чтобы позволить программам [Java](https://ru.bmstu.wiki/Java) работать одинаково на всех из них.

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

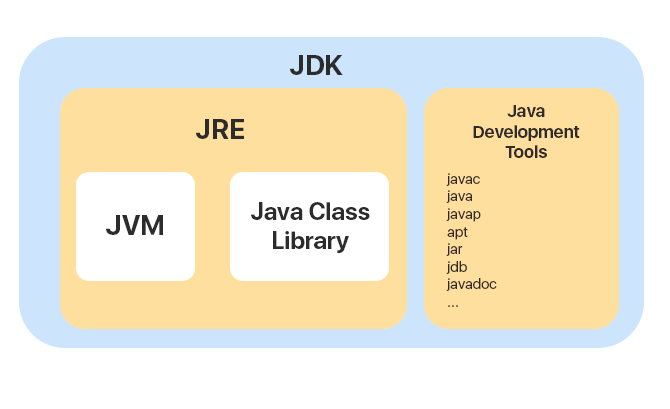
Ответ:

Все используемые среды java предполагают, что исходный код класса с модификатором public хранится в файле с точно таким же именем, как и имя класса, и расширением .java. Несоблюдение этого соглашения может стать причиной многих проблем, которые выявятся во время компиляции.

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

Ответ:

JVM — это программа, предназначенная для выполнения байт-кода. Первое преимущество JVM — это принцип “Write once, run anywhere”. Он означает, что приложение, написанное на Java, будет работать одинаково на всех платформах. Это является большим преимуществом JVM и самой Java.

Java Development Kit — комплект разработчика приложений на языке Java. Он включает в себя Java Development Tools и среду выполнения Java — JRE (Java Runtime Environment).

Среда выполнения JRE — это пакет всего необходимого для запуска скомпилированной Java-программы. Включает в себя виртуальную машину JVM и библиотеку классов Java — Java Class Library.

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

Ответ:

Открыть командную cтроку. В рабочем каталоге нужно скомпилировать программу, для чего нужно ввести **javac** затем через пробел адрес к файлу (после текущего каталога)\ имя файла с расширением (напр., **javac by\jonline\lec05\main\Main.java**) и нажимаем enter. После этого можно запускать программу с помощью команды **java** затем через пробел имя (с полным указанием пакета) класса без расширения (напр., **java** **by.jonline.lec05.main.Main**).

Переменные среды - это пары ключ / значение, которые определяют конфигурации для различных программ в операционной системе. Обычно эти переменные используются, когда мы хотим выполнить программу с помощью команды, не переходя или не записывая полный путь к программе. Например, если мы хотим запустить Java-программу где угодно, придется сначала перейти в то место, где находится Java, через командную строку или терминал, а затем мы сможем использовать Java. Но если мы укажем путь к Java в переменных среды, мы должны использовать имя программы или прикрепленную к ней команду, и программа будет выполнена.

1. Перечислите атрибуты доступа, объясните их действие.

Ответ:

**private** – наиболее строгий модификатор доступа. Он ограничивает видимость данных и методов пределами одного класса;

**protected** – поля и методы, обозначенные модификатором доступа protected, будут видны:

-в пределах всех классов, находящихся в том же пакете, что и наш;

-в пределах всех классов-наследников нашего класса.;

**default (package visible)** – видны в пределах всех классов, находящихся в том же пакете, что и наш;

**public** – Части кода, помеченные модификатором public, предназначаются для конечного пользователя.

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

Ответ:

Пакеты, по сути, являются файловой и логической структурой связей классов в мире java. Очень схоже с файловой системой компьютера. На уровне файловой системы пакеты это и есть папки, в которых лежат другие папки (подпакеты) и классы. Пакеты необходимы для разделения доступа классов друг к другу.

Пакеты всегда пишутся строчными буквами. В отличие от классов, где первая буква обычно является заглавной. Чтобы избежать создания пакетов с такими же именами, как у других общедоступных, рекомендуется называть url сайта в обратном порядке. Если в домене используются цифры или спецсимволы, зарезервированные слова Джава - то может использоваться нижнее подчеркивание.

Как создать пакет в Java? Во время создания пакета вы должны выбрать ему имя и включить оператор package вместе с этим именем поверх каждого исходного файла, который содержит классы, интерфейсы, перечисления и типы аннотаций, которые вы хотите включить в пакет. Оператор package должен быть первой строкой в исходном файле. Может быть только один запрос package в каждом исходном файле, и он применяется ко всем типам в этом файле. Если оператор package не использован, тогда классы, интерфейсы, перечисления и типы аннотаций будут помещены в текущий пакет по умолчанию. Чтобы скомпилировать программы на Java с операторами package, то вам нужно использовать опцию –d, как показано ниже.

javac -d Destination\_folder file\_name.java

1. *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?*

Ответ:

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.

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

Ответ:

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.

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

Ответ:

Различают 8 примитивных типов в Java:

- целочисленные типы:

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

byte a;

a = -6;

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

short x;

x = 120;

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

int i, o;

i = 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

}

1. 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.

Ответ:

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 - 8 бит.

boolean – 8 бит в массивах, 32 бит вне массива;

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 иногда необходимо преобразование одного типа данных в другой.

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

Ответ:

Преобразование типов в Java бывает двух видов: неявное (расширение) и явное (сужение).

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.

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

Ответ:

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. Логические;
3. Символьные;
4. Строковые.

1. Числовые литералы

1.1. Целочисленные литералы

Типы целочисленных литералов в Java:

десятиричные (4567);

восьмеричные - начинаются с 0 и используют числа от 0 до 7 (01248);

шестнадцатеричные - создаются используя следующие символы [0-9a-fA-F], Должны начинаться с 0x или 0X, Разрешается использовать до 16 символов в шестнадцатеричных числах, не учитывая префикс 0x и необязательный суффикс L (0x654AB5);

двоичные - начиная с Java 7 - Для определения двоичного литерала, добавьте префикс 0b или 0B к числу. Используются числа 0 и 1. (0B01100111)

Все целочисленные литералы представляют значения int. Если значение литерала лежит в диапазоне byte, short или char, то его можно присвоить переменной этого типа без приведения типов. Для создания литерала типа long, необходимо явно указать компилятору, дополнив литерал буквой 'l' или 'L':

long a1 = 220433L;

long a2 = 0x3FFl;

1.2 Литералы с плавающей точкой

Всем литералам с плавающей точкой по умолчанию присваивается тип double. Чтобы создать литерал типа float, нужно после литерала указать букву 'f' или 'F'. Например:

float f1 = 56.45455; // Ошибка компиляции

float f2 = 343434.454563F;

float f3 = 78.45f;

Можно добавить символ 'D' или 'd' к double литералу, но это не обязательно.

Существует так называемая научная нотация записи вещественных чисел:

double d6 = 1.5e14;

Значение этого числа равно 1.5 умножить на 10 в четырнадцатой степени. Может использоваться строчная или прописная буква - 'e', 'E'. Также степень может содержать знак + или -. Например:

double d7 = 1.5E+14;

double d8 = 1.5e-14;

1.3. Подчеркивание в числовых литералах

Начиная с Java 7, добавлена возможность использовать любое количество символов подчеркивания для разделения групп цифр, что улучшает читабельность.

Подчеркивание может разделять только цифры! Нельзя использовать подчеркивания в следующих местах:

* В начале или конце числа
* Рядом с десятичной точкой в числе с плавающей точкой
* Перед 'F', 'f', 'l' или 'L' суффиксами
* Для разделения символов префиксов.

2. Логические Boolean литералы

Значение литерала типа boolean может быть определено как true или false:

boolean b1 = true; // Правильно

boolean b2 = 0; // Не правильно

3. Символьные литералы

Для представления символов типа char в Java используется кодировка в Unicode, определяющая полный набор международных символов на всех известных языках мира.

Символьные литералы могут быть представлены в виде символа заключенного в одинарные кавычки:

char c1 = 'n';

char c2 = '#';

Существует также механизм для непосредственного ввода значения символа в восьмеричной или шестнадцатеричной форме. Для ввода значений в восьмеричной форме служит символ обратной косой черты, за которым следует трехзначное число. Например:

char letterO = '\u004F'; // Буква 'O' в шестнадцатеричной форме

char letter\_a = '\141'; // Буква 'a' в восьмеричной форме

Есть возможность присвоения числового литерала символьному типу. Числовой литерал должен находиться в диапазоне от 0 до 65535. Например:

char a1 = 0x675; // шестнадцатеричный целочисленный литерал

char a2 = 345; // десятеричный целочисленный литерал

char a3 = (char)80000; // Делается приведение типов, т.к. 80000 //превышает диапазон char типа

Символ обратной косой черты может быть использован для обозначения символов не имеющих точного написания, например символ перевода строки, табуляции, backspace, двойные и одинарные кавычки:

char c1 = '\'';

char c2 = '\n';

4. Строковые литералы

Для обозначения строковых литералов последовательность символов заключается в двойные кавычки.

Управляющие символы и восьмеричная или шестнадцатеричная форма записи, определенные для символьных литералов, действуют точно так же и в строковых литералах.

Строки могут располагаться только на одной строке исходного кода, нельзя открывающую кавычку поставить на одной строке, а закрывающую — на следующей.

String str1 = "MyProgram";

String str2 = "first line\nsecond line";

String str3 = "\"This is in quotes\"";

String str4 = "\u004F letter";//O letter

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

Ответ:

Присвоение переменной значения константы, другой переменной или выражения (переменных и/или констант, разделенных знаками операций), называется операцией присваивания и обозначается знаком "=", например:

x = 3; y = x; z = x;

Бинарные знаки операций подразделяются на операции с числовым результатом и операции сравнения, результатом которых является булевское значение.

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

* унарный минус "-" – меняет знак числа или выражения на противоположный;
* унарный плюс "+" – не выполняет никаких действий над числом или выражением;
* побитовое дополнение "~" (только для целых) – инвертирует все биты поля числа (меняет 0 на 1 и 1 на 0);
* 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 числовыми 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.

Побитовые операции

Побитовые операции рассматривают исходные числовые значения как поля битов и выполняют над ними следующие действия:

* установка бита в i-ой позиции поля результата в 1, если оба бита в i-ых позициях операндов равны 1, или в 0 в противном случае – побитовое И ("&");
* установка бита в i-ой позиции поля результата в 1, если хотя бы один бит в i-ых позициях операндов равен 1, или в 0 в противном случае – побитовое ИЛИ ("|");
* установка бита в i-ой позиции поля результата в 1, если биты в i-ых позициях операндов не равны друг другу, или в 0 в противном случае – побитовое исключающее ИЛИ ("^");
* сдвиг влево битов поля первого операнда на количество битов, определяемое вторым операндом (бит знака числа при этом не меняется) – побитовый сдвиг влево с учетом знака "<<";
* сдвиг вправо битов поля первого операнда на количество битов, определяемое вторым операндом (бит знака числа при этом не меняется) – побитовый сдвиг вправо с учетом знака ">>";
* сдвиг вправо битов поля первого операнда на количество битов, определяемое вторым операндом (бит знака числа при этом также сдвигается) – побитовый сдвиг вправо без учета знака ">>>".

Комбинированные операции

В Java для бинарных арифметических операций можно использовать комбинированные (составные) знаки операций:

идентификатор операция = выражение

Это эквивалентно следующей операции: идентификатор = идентификатор операция выражение

Пример:

Выражение x += b означает x = x + b.

Выражение x -= b означает x = x - b.

Выражение x \*= b означает x = x \* b.

Выражение x /= b означает x = x / b.

Выражение x %= b означает x = x % b.

Выражение x &= b означает x = x & b.

Выражение x |= b означает x = x | b.

Выражение x ^= b означает x = x ^ b.

Выражение x <<= b означает x = x << b.

Выражение x >>= b означает x = x >> b.

Выражение x >>>= b означает x = x >>> b.

Операции сравнения

В Java определены следующие операции сравнения:

"==" (равно), "!=" (не равно),

">" (больше), ">=" (больше или равно),

"<" (меньше) "<=" (меньше или равно)

имеют два операнда и возвращают булевское значение, соответствующее результату сравнения (false или true).

Булевские операции

Булевские операции выполняются над булевскими переменными и их результатом также является значение типа boolean. В Java определены следующие булевские операции:

* отрицание "!" – замена false на true, или наоборот;
* операция И "&" – результат равен true, только, если оба операнда равны true, иначе результат – false;
* операция ИЛИ "|" – результат равен true, только, если хотя бы один из операндов равен true, иначе результат – false.
* операция исключающее ИЛИ "^" – результат равен true, только, если операнды не равны друг другу, иначе результат – false.

Операции "&", "|" и "^" можно, также как и соответствующие побитовые операции использовать в составных операциях присваивания: "&=", "|=" и "^=" Кроме того, к булевским операндам применимы операции "==" (равно) и "!=" (не равно).

В Java определены еще две булевские операции: вторые версии булевских операций И и ИЛИ, известные как укороченные (short-circuit) логические операции: укороченное И "&&" и укороченное ИЛИ "||". При использовании этих операций второй операнд вообще не будет вычисляться, что полезно в тех случаях, когда правильное функционирование правого операнда зависит от того, имеет ли левый операнд значение true или false.

Условная операция

Условная операция записывается в форме выражение -1?выражение-2:выражение-3. При этом сначала вычисляется выражение выражение-1, которое должно дать булевское значение, а затем, если выражение-1 имеет значение true, вычисляется и возвращается выражение-2 как результат выполнения операции, либо (если выражение-1 имеет значение false), вычисляется и, как результат выполнения операции, возвращается выражение-3. Пример условной операции:

x=n>1?0:1;

Переменной x будет присвоено значение 0, если n>1 (выражение n>1 имеет значение true) или 1, если n≤1 (выражение n>1 имеет значение false).

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

Ответ:

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.

1. 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.

Ответ:

Каждая переменная в 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.

1. Поясните, что такое автоупаковка и автораспаковка.

Ответ:

Автоупаковка и распаковка это функция автоматического преобразования примитивных типов в объектные и наоборот.

Пример Автоупаковки:

Integer integer = 9;

Пример автораспаковки:

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.

1. 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.

Ответ:

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;

++b; // результат 6 (до вычислений)

b++; // результат 6 (после вычислений)

-- - декремент (отнимает 1):

int b = 5;

--b; // результат 4 (до вычислений)

b--; // результат 4 (после вычислений)

+= - 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 (При каждом сдвиге вправо выполняется деление на два с отбрасыванием любого остатка)

<< - сдвиг влево: byte x = (byte) (100 << 5); // результат -128 (При каждом сдвиге влево выполняется умножение числа на 2 с учётом количества ячеек-битов)

>>> - 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

**Булевые 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 | | | |

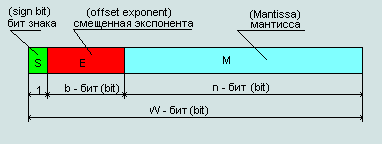
Если используются операторы одинакового уровня, то приоритет будет у того оператора, который находится левее.

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

Ответ:

 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

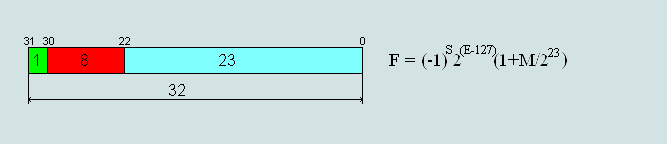
E - смещенная экспонента двоичного числа; exp2 = E - (2(b-1) - 1) - экспонента двоичного нормализованного числа с плавающей точкой (2(b-1) -1) - заданное смещение экспоненты (в 32-битном IEEE754 оно равно +127)

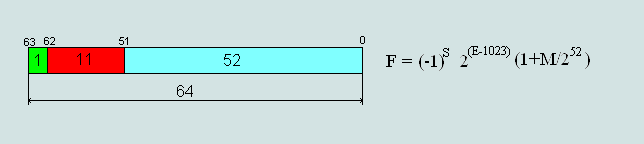
M - остаток мантиссы двоичного нормализованного числа с плавающей точкой

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.

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

Ответ:

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);

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

Ответ:

**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. команада будет выполнена как минимум 1 раз даже при false

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.

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

Ответ:

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