**Хорошо изучите материал по темам (будьте готовы отвечать на вопросы):**

**Зарезервированные слова в Javа**: Зарезервированные ключевые слова—это специальные идентификаторы, которые в языке Java используются для того, чтобы идентифицировать встроенные типы, модификаторы и средства управления выполнением программы. На сегодняшний день в языке Java имеется 59 зарезервированных слов. Эти ключевые слова совместно с синтаксисом операторов и разделителей входят в описание языка Java. Они могут применяться только по назначению, их нельзя использовать в качестве идентификаторов для имен переменных, классов или методов.(abstract, boolean, break, byte, byvalue, case, cast, catch, char, class, const, continue, default do, double, else, extends, false, final, finally, float, for, future, generic, goto, if, implements, import, inner, instanceof, int, interface, long, native, new, null, operator, outer, package, private, protected, public, rest, return, short, static, super, switch, syncronized, this, throw , throws, transient, true, try, var, void, volatile, while.

**Зарезервированные имена методов Java**- Отметим, что слова byvalue, cast, const, future, generic, goto, inner, operator, outer, rest, var зарезервированы в Java, но пока не используются Кроме этого, в Java есть зарезервированные имена методов (clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait)

**Идентификаторы**

**Примитивные типы (primitive types)** целочисленные типы byte, short, int, long, char;  
**Типы данных в формате с плавающей точкой** float, double; булевый (булевский, логический) тип boolean;   
**Ссылочные типы** (Object, Float, Boolean, Collection)

Идентификаторы используются для именования классов, методов и переменных. В качестве идентификатора может использоваться любая последовательность строчных и прописных букв, цифр и символов \_ (подчеркивание) и $ (доллар). Идентификаторы не должны начинаться с цифры, чтобы транслятор не перепутал их с числовыми литеральными константами, которые будут описаны ниже. Java—язык, чувствительный к регистру букв.

**Литералы (целые, с плавающей точкой, логические, символьные, строчные) и константы**

Константы в Java задаются их литеральным представлением. Целые числа, числа с плавающей точкой, логические значения, символы и строки можно располагать в любом месте исходного кода.   
**Целые литералы**:Целые числа—это тип, используемый в обычных программах наиболее часто. Любое целочисленное значение(1, 2, 3, 42)—это целый литерал. Кроме десятичных, в качестве целых литералов могут использоваться также числа с основанием 8 и 16—восьмеричные и шестнадцатиричные. Java распознает восьмеричные числа по стоящему впереди нулю. Шестнадцатиричная константа различается по стоящим впереди символам нуль-х (0х или 0Х). Диапазон значений шестнадцатиричной цифры—0.. 15, причем в качестве цифр для значений 10.. 15 используются буквы от А до F (или от а до f).   
Целые литералы являются значениями типа int или типа long. К числам с любым из названных выше оснований вы можете приписать справа строчную или прописную букву L, указав таким образом, что данное число относится к типу long.

**Литералы с плавающей точкой:** Числа с плавающей точкой представляют десятичные значения, у которых есть дробная часть. Их можно записывать либо в обычном, либо экспоненциальном форматах. В обычном формате число состоит из некоторого количества десятичных цифр, стоящей после них десятичной точки, и следующих за ней десятичных цифр дробной части(2.0, 3.14159 и .6667). В экспоненциальном формате после перечисленных элементов дополнительно указывается десятичный порядок. Порядок определяется положительным или отрицательным десятичным числом, следующим за символом Е или е(6.022е23, 314159Е-05, 2е+100). В Java числа с плавающей точкой по умолчанию рассматриваются, как значения типа double. Если вам требуется константа типа float, справа к литералу надо приписать символ F или f. Если вы любитель избыточных определений—можете добавлять к литералам типа double символ D или d.

**Логические литералы**

У логической переменной может быть лишь два значения—true (истина) и false (ложь). Логические значения true и false не преобразуются ни в какое числовое представление. Ключевое слово true в Java не равно 1, a false не равно 0. В Java эти значения могут присваиваться только переменным типа boolean либо использоваться в выражениях с логическими операторами.

**Символьные литералы**

Символы в Java—это индексы в таблице символов UNICODE. Они представляют собой 16-битовые значения, которые можно преобразовать в целые числа и к которым можно применять операторы целочисленной арифметики, например, операторы сложения и вычитания. Символьные литералы помещаются внутри пары апострофов (' '). Все видимые символы таблицы ASCII можно прямо вставлять внутрь пары апострофов:—'a', 'z', '@'. Для символов, которые невозможно ввести непосредственно, предусмотрено несколько управляющих последовательностей.

**Строчные литералы:**Строчные литералы в Java —это произвольный текст, заключенный в пару двойных кавычек (""). Хотя строчные литералы в Java реализованы весьма своеобразно (Java создает объект для каждой строки), внешне это никак не проявляется. Примеры строчных литералов: "Hello World!"; "две\строки; \ А это в кавычках\"". Строчные литералы в Java должны начинаться и заканчиваться в одной и той же строке исходного кода.

***Константа* —** это именованная ячейка памяти, способная хранить данные, которые потом изменяться не будут.в Java для хранения данных можно использовать **константы**. В отличие от переменных константам можно присвоить значение только один раз. Константа объявляется также, как и переменная, только вначале идет ключевое слово **final**

**Управляющие символы   
Управля́ющие си́мволы —**символы в кодировке, которым не приписано графическое представление, но которые используются для управления устройствами, организации передачи данных и других целей.

**Управляющая последовательность Описание**

\ddd Восьмеричный символ (ddd)

\uxxxx Шестнадцатиричный символ UNICODE (xxxx)

\' Апостроф

\" Кавычка

\\ Обратная косая черта

\r Возврат каретки (carriage return)

\n Перевод строки (line feed, new line)

\f Перевод страницы (form feed)

\t Горизонтальная табуляция (tab)

\b Возврат на шаг (backspace)

\t горизонтальная табуляция

\u начало кодировки символа Unicode

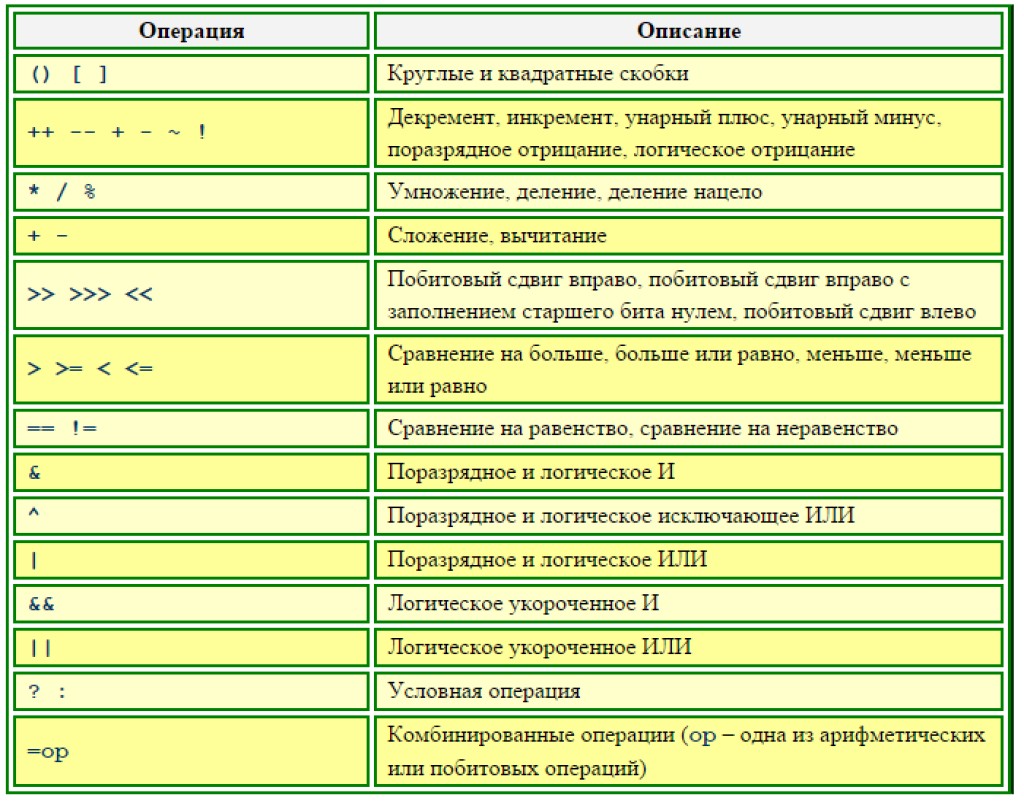
\u0030 — \u0039 — цифры ISO-LATIN от 0 до 9;

\u0024 — знак доллара $;

\u0041 — \u005a — буквы от A до Z;

\u0061 — \u007a — буквы от a до z.

**Операции и примитивные типы данных (целые типы, числа с плавающей точкой, символы и кодировки)**



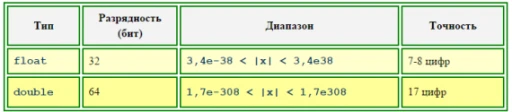
в Java определены следующие примитивные типы: целые типы; вещественные типы; булевский тип. За оператором объявления примитивной переменной может следовать оператор инициализации "=", с помощью которого созданной переменной присваи­вается начальное значение.

**1. Целые типы переменных**

**Целые** типы различаются по размеру отведенной для них памяти.



## 2. Вещественные типы переменных



## 3. Булевский тип переменных

Переменные булевского типа (логические переменные) могут принимать одно из двух значений: «истина» или «ложь» и используются в языках программирования в операциях отношения (сравнения) и логических операциях. Переменные *булевского типа* в Java задаются с помощью ключевого слова boolean и могут иметь лишь одно из двух значений: *true* или *false*, например

**Разделители.** Лишь несколько групп символов, которые могут появляться в синтаксически правильной Java-программе, все еще остались неназваннами. Это—простые разделители, которые влияют на внешний вид и функциональность программного кода.

|  |  |  |
| --- | --- | --- |
| **Символы** | **Название** | **Для чего применяются** |
| ( ) | круглые скобки | Выделяют списки параметров в объявлении и вызове метода, также используются для задания приоритета операций в выражениях, выделения выражений в операторах управления выполнением программы, и в операторах приведения типов. |
| { } | фигурные скобки | Содержат значения автоматически инициализируемых массивов, также используются для ограничения блока кода в классах, методах и локальных областях видимости. |
| [ ] | квадратные скобки | Используются в объявлениях массивов и при доступе к отдельным элементам массива. |
| ; | точка с запятой | Разделяет операторы. |
| , | запятая | Разделяет идентификаторы в объявлениях переменных, также используется для связи операторов в заголовке цикла for. |
| . | точка | Отделяет имена пакетов от имен подпакетов и классов, также используется для отделения имени переменной или метода от имени переменной. |

**Переменные.Переменная**—это основной элемент хранения информации в Java-программе. Переменная характеризуется комбинацией идентификатора, типа и области действия. В зависимости от того, где вы объявили переменную, она может быть локальной, например, для кода внутри цикла for, либо это может быть переменная экземпляра класса, доступная всем методам данного класса. Локальные области действия объявляются с помощью фигурных скобок.

**Объявление переменной**

**Основная форма объявления переменной такова:**

тип идентификатор [ = значение] [, идентификатор [ = значение 7...];

Тип—это либо один из встроенных типов, то есть, byte, short, int, long, char, float, double, boolean, либо имя класса или интерфейса.. Ниже приведено несколько примеров объявления переменных различных типов. Переменные, для которых начальные значения не указаны, автоматически инициализируются нулем.

**Операторы Оператор—**это нечто, выполняющее некоторое действие над одним или двумя аргументами и выдающее результат. Синтаксически операторы чаще всего размещаются между идентификаторами и литералами.

|  |  |  |  |
| --- | --- | --- | --- |
| + | += | - | -= |
| \* | \*= | / | /= |
| | | |= | ^ | ^= |
| & | &= | % | %= |
| > | >= | < | <= |
| ! | != | ++ | — |
| >> | >>= | << | <<= |
| >>> | >>>= | && | | | |
| == | = | ~ | ?: |
|  | instanceof | [ ] |  |
| **Арифметические операторы используются** для вычислений так же как в алгебре . Допустимые операнды должны иметь числовые типы.   |  |  |  |  | | --- | --- | --- | --- | | **Оператор** | **Результат** | **Оператор** | **Результат** | | + | Сложение | + = | сложение с присваиванием | | - | вычитание (также унарный минус) | -= | вычитание с присваиванием | | \* | Умножение | \*= | умножение с присваиванием | | / | Деление | /= | деление с присваиванием | | % | деление по модулю | %= | деление по модулю с присваиванием | | ++ | Инкремент | — | декремент |   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. В таблице приведена сводка таких операторов. Операторы битовой арифметики работают с каждым битом как с самостоятельной величиной.   |  |  |  |  | | --- | --- | --- | --- | | **Оператор** | **Результат** | **Оператор** | **Результат** | | ~ | побитовое унарное отрицание (NOT) |  |  | | & | побитовое И (AND) | &= | побитовое И (AND) с присваиванием | | | | побитовое ИЛИ (OR) | |= | побитовое ИЛИ (OR) с присваиванием | | ^ | побитовое исключающее ИЛИ (XOR) | ^= | побитовое исключающее ИЛИ (XOR) с присваиванием | | >> | сдвиг вправо | >> = | сдвиг вправо с присваиванием | | >>> | сдвиг вправо с заполнением нулями | >>>= | сдвиг вправо с заполнением нулями с присваиванием | | << | сдвиг влево | <<= | сдвиг влево с присваиванием | |  |  |  |

**Классы оболочки (ссылочные типы данных)**

Классы-оболочки Java являются Объектным представлением восьми примитивных типов в Java. Все классы-оболочки в Java являются неизменными и final. Начиная с Java 5 [автоупаковка и распаковка](https://javadevblog.com/avtoupakovka-i-raspakovka-v-java-na-primere.html) позволяет легко конвертировать примитивные типы в их соответствующие классы-оболочки и наоборот.

|  |  |  |
| --- | --- | --- |
| **Примитивный тип** | **Класс-обертка** | **Аргументы** |
| byte | Byte | byte или String |
| short | Short | short или String |
| int | Integer | int или String |
| long | Long | long или String |
| float | Float | float, double или String |
| double | Double | double или String |
| char | Character | char |
| boolean | Boolean | boolean или String |

**Зачем нужны классы-оболочки в Java?**

Разработчиками языка Java было принято очень умное решение отделить примитивные типы и классы-оболочки, указав при этом следующее:

* Используйте классы-обертки, когда работаете с коллекциями.
* Используйте примитивные типы для того, чтобы ваши программы были максимально просты.

Еще одним важным моментом является то, что примитивные типы **не могут** быть **null**, а классы-оболочки — **могут**.

Также классы-оболочки могут быть использованы для достижения полиморфизма.

**Массивы (многомерные массивы)**

**Массив** — это упорядоченное множество элементов одного типа, примитивного или ссылочного. Массив, элементами которого являются другие массивы, то есть массив массивов, называется двумерным. В обобщённом виде многомерные массивы в Java выглядят так:

Data\_type[dimension1][dimension2][]..[dimensionN] array\_name = **new** data\_type[size1][size2]….[sizeN];

Где Data\_type — это тип элементов в массиве. Может быть примитивным или ссылочным (классом). Количество пар скобок с dimension внутри — размерность массива (в нашем случае — N). array\_name — название массива size1...sizN — количество элементов в каждом из измерений массива. Объявление многомерных массивов:

**int**[][] twoDimArray; //двумерный массив

String[][][] threeDimArray; //трёхмерный массив

**double**[][][][][] fiveDimArray; // пятимерный массив

**Тип String.** String — это класс в Java, который прописан в пакете java.lang. Это не примитивный тип данных, как int и long. Класс String представляет строковый набор символов. Мы можем создавать объекты используя оператор new, как и для любого другого класса в Java или мы можем использовать двойные кавычки для создания объекта String. Также есть несколько конструкторов класса String для получения строки из массива символов, массива байтов, а также используя StringBuffer или StringBuilder.

String str = **new** String("abc");

String str1 = "abc";

Когда мы создаем строку используя двойные кавычки, виртуальная машина Java ищет в пуле строк другую строку с таким же значением. Если строка найдена, то возвращается только ссылка на существующий объект класса String, иначе создается новый объект с полученным значением, и сохраняется в пул. Когда мы используем оператор new, виртуальная машина создает объект String, но не хранит его в пуле строк. Мы можем использовать метод intern() для сохранения строки в пуле строк, или получения ссылки, если такая строка уже находится в пуле.

**Консольный ввод-вывод .**Для создания потока вывода в класс System определен объект **out**. В этом объекте определен метод **println**, который позволяет вывести на консоль некоторое значение с последующим переводом курсора консоли на следующую строку.System.out.println("Hello world!");

При необходимости можно и не переводить курсор на следующую строку. В этом случае можно использовать метод **System.out.print()**, который аналогичен println за тем исключением, что не осуществляет перевода на следующую строку

### Для получения ввода с консоли в классе System определен объект **in**. Однако непосредственно через объект System.in не очень удобно работать, поэтому, как правило, используют класс **Scanner**, который, в свою очередь использует System.in. Например, напишем маленькую программу, которая осуществляет ввод чисел:

Scanner in = new Scanner(System.in);

        System.out.print("Input a number: ");

        int num = in.nextInt();

        System.out.printf("Your number: %d \n", num);

        in.close();

Так как класс Scanner находится в пакете **java.util**, то мы вначале его импортируем с помощью инструкции import java.util.Scanner. Для создания самого объекта Scanner в его конструктор передается объект System.in. После этого мы можем получать вводимые значения.Класс Scanner имеет еще ряд методов, которые позволяют получить введенные пользователем значения:

* **next()**: считывает введенную строку до первого пробела
* **nextLine()**: считывает всю введенную строку
* **nextInt()**: считывает введенное число int
* **nextDouble()**: считывает введенное число double
* **nextBoolean()**: считывает значение boolean
* **nextByte()**: считывает введенное число byte
* **nextFloat()**: считывает введенное число float
* **nextShort()**: считывает введенное число short

**Документирование кода**

**javadoc** — это генератор документации в HTML-формате из комментариев исходного кода Java и определяет стандарт для документирования классов Java. Для создания доклетов и тэглетов, которые позволяют программисту анализировать структуру Java-приложения, **javadoc** также предоставляет API. В каждом случае комментарий должен находиться перед документируемым элементом.

При написании комментариев к кодам Java используют три типа комментариев :

// однострочный комментарий;

/\* многострочный комментарий \*/

/\*\* комментирование документации \*/

|  |  |  |
| --- | --- | --- |
| **Дескриптор** | **Применение** | **Описание** |
| @author | Класс, интерфейс | Автор |
| @version | Класс, интерфейс | Версия. Не более одного дескриптора на класс |
| @since | Класс, интерфейс, поле, метод | Указывает, с какой версии доступно |
| @see | Класс, интерфейс, поле, метод | Ссылка на другое место в документации |
| @param | Метод | Входной параметр метода |
| @return | Метод | Описание возвращаемого значения |
| @exception имя\_класса описание | Метод | Описание исключения, которое может быть послано из метода |
| @throws имя\_класса описание | Метод | Описание исключения, которое может быть послано из метода |
| @deprecated | Класс, интерфейс, поле, метод | Описание устаревших блоков кода |
| {@link reference} | Класс, интерфейс, поле, метод | Ссылка |
| {@value} | Статичное поле | Описание значения переменной |

**Code convention**

[Magnum news - Соглашения по оформлению кода Java (Java Code Conventions) (magnumblog.space)](https://www.magnumblog.space/java/131-translating-java-code-conventions)

**Поменяйте уровень языка, разберитесь с вкладками Libraries, Artifacts и т.д.**

Артефакт - это сборка активов вашего проекта, которые вы собрали для тестирования, развертывания или распространения вашего программного решения или его части. Примерами являются набор скомпилированных классов Java или приложения Java, упакованных в архив Java, веб-приложение в виде структуры каталогов или архива веб-приложений и т. д.

Библиотеки же являются зависимостями, которые используются вашей программой в работе, и сделаны кем то другим.

Вы можете сколько угодно работать над проектом не генерируя ни одного артефакта. Но вряд ли у вас что то получиться, если вы не подключили нужные вам библиотеки, или подключили неподходящую версию.

**Фасет** (**facet**) - это специальный дочерний компонент, к которому можно получить доступ с помощью метода UIComponent.getFacet. Этот метод может быть переопределен для пользовательских компонентов; данное обстоятельство позволяет иначе обрабатывать компоненты в **фасете**.

**Каково их назначение(**TODO)**?**

Смысл в них в том, что они могут особым образом обрабатываться в IDE. Например, в IDEA они собираются на отдельной вкладке,

Во время написания кода очень часто возникает ситуация, когда вы заметили какой-либо недочёт в коде, который не настолько критичен, чтобы править его прямо сейчас, но очень желательно исправить его в будущем. Чтобы потом о них не забыть, в таких местах можно оставлять комментарии, предваряя их токеном TODO с двоеточием:

[**Аннотация (Java)**](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%BD%D0%BE%D1%82%D0%B0%D1%86%D0%B8%D1%8F_(Java))

**Java**-**аннотация** — в языке **Java** специальная форма синтаксических метаданных, которая может быть добавлена в исходный код. **Аннотации** используются для анализа кода, компиляции или выполнения. Аннотируемы пакеты, классы, методы, переменные и параметры. Выглядит как @ИмяАннотации, предваряющее определение переменной, параметра, метода, класса, пакета.