**Лекция 3**

**3.1 Основные концепции OOPS**

**Класс**

Класс представляет собой группу похожих объектов. Это только логический компонент, а не физический объект. Например, если у вас есть класс под названием «Дорогие автомобили», у него могут быть такие объекты, как Mercedes, BMW, Toyota и т. Д. Его свойства (данные) могут быть ценой или скоростью этих автомобилей. В то время как методы могут быть выполнены с этими автомобилями, вождение, задний ход, торможение и т. д.

**Объект**

Объект может быть определен как экземпляр класса, и в программе может быть несколько экземпляров класса. Объект содержит как данные, так и функцию, которая работает с данными. Например – стул, велосипед, маркер, ручка, стол, машина.

**Наследование**

Наследование – это концепция OOPS, в которой один объект приобретает свойства и поведение родительского объекта. Это создает **родительско-дочерние** отношения между двумя классами. Он предлагает надежный и естественный механизм для **организации и структуры любого программного обеспечения**.

**Полиморфизм**

Полиморфизм относится к способности **переменной**, **объекта** или функции принимать несколько форм. Например, в русском (да и в других языках) понимание значения слова зависит от других слов, используемых вместе с ним. То же самое относится и к полиморфизму в ООП.

Задание на лету – придумайте примеры из русского языка.

**Абстракция**

Абстракция является актом представления основных функций без включения деталей фона. Это метод создания нового типа данных, который подходит для конкретного приложения. Например, во время вождения автомобиля вам не нужно беспокоиться о его внутренней работе. Здесь вам просто нужно заботиться о таких деталях, как руль, шестерни, акселератор.

**Инкапсуляция**

Инкапсуляция – это ООП метод упаковки данных и кода. В этой концепции OOPS переменные класса всегда скрыты от других классов. **Доступ к нему возможен только с использованием методов их текущего класса**. Например – в школе ученик не может существовать без класса.

**Ассоциация**

Ассоциация – это связь между двумя объектами. Это определяет разнообразие между объектами. В этой концепции ООП все объекты имеют отдельный жизненный цикл, и у них нет владельца. Например, многие ученики могут общаться с одним учителем, а один ученик также может общаться с несколькими учителями.

<https://ru.stackoverflow.com/questions/1066615/Ассоциация-vs-Агрегация-в-чем-разница>

<https://www.examclouds.com/ru/java/java-core-russian/class-relations>

**Агрегация**

**В этой технике все объекты (*в общем смысле*) имеют отдельный жизненный цикл**. Это означает, что объекты класса относительно самостоятельны по отношению к классу. Тем не менее, существует владение, так что дочерний объект не может принадлежать другому родительскому объекту. Например, рассмотрим **класс** «кафедра» и его объект «учитель». Здесь, один учитель не может принадлежать нескольким кафедрам, но даже если мы удалим кафедру, объект «учитель» не будет уничтожен.

Абстракция означает использование простых вещей для описания чего-то сложного. Например, мы все знаем, как пользоваться телевизором, но в тоже время нам не нужно обладать знаниями о том, как он работает чтобы смотреть его. **В Java под абстракцией подразумеваются такие вещи, как объекты**, **классы** и **переменные**, которые в свою очередь лежат в основе более сложного кода. Использование данного принципа позволяет избежать сложности при разработке ПО.

**Агрегирование также является отношением “имеет-а”**. Что отличает его от композиции, так это то, что оно не предполагает владения. В результате жизненные циклы объектов не связаны: каждый из них может существовать независимо друг от друга.

Например, автомобиль и его колеса. **Мы можем убрать колеса, и они все равно будут существовать.** Мы можем установить другие (уже существующие) колеса или установить их на другую машину, и все будет работать просто отлично.

Конечно, автомобиль без колес или с отсоединенным колесом не будет таким полезным, как автомобиль с включенными колесами. Но именно поэтому эта связь существовала в первую очередь: **собрать части в более крупную конструкцию, которая способна на большее, чем ее части**.

Поскольку агрегирование не предполагает владения, **член не обязательно должен быть привязан только к одному контейнеру**. Например, треугольник состоит из сегментов. Но треугольники могут иметь общие сегменты в качестве своих сторон.

<https://translated.turbopages.org/proxy_u/en-ru.ru.da292b1a-63df90d0-7abeaea0-74722d776562/https/www.baeldung.com/java-composition-aggregation-association> !!!!!!

**Композиция**

Композиция – это **специализированная форма агрегации**. Это также называется “смертью” отношений. **Дочерние объекты не имеют своего жизненного цикла, поэтому при удалении родительского объекта все дочерние объекты также удаляются автоматически**. Для этого давайте возьмем пример дома и комнаты. Любой дом может иметь несколько комнат. Одна комната не может стать частью двух разных домов. Так что, если вы удалите дом, номер также будет удален.

***Композиция*** - это тип отношений “принадлежит”. Это означает, что один из объектов представляет собой логически большую структуру, которая содержит другой объект. Другими словами, это часть или член другого объекта.

В качестве альтернативы, **мы часто называем это отношением “имеет-а” (в отличие от отношения** “есть-а”, которое является [наследованием](https://translated.turbopages.org/proxy_u/en-ru.ru.da292b1a-63df90d0-7abeaea0-74722d776562/https/www.baeldung.com/java-inheritance)).

Например, комната принадлежит зданию, или, другими словами, в здании есть комната. Итак, в принципе, называем ли мы это “принадлежит” или “имеет”, это только вопрос точки зрения.

Композиция - это **сильный** вид отношений “имеет-а”, потому что содержащий объект владеет им. Таким образом, **жизненные циклы объектов связаны. Это означает, что если мы уничтожим объект owner, его члены также будут уничтожены вместе с ним.** Например, комната уничтожается вместе со зданием в нашем предыдущем примере.

Обратите внимание, что это не означает, что содержащий объект не может существовать без какой-либо из его частей. Например, мы можем снести все стены внутри здания, следовательно, уничтожить комнаты. Но здание все равно будет существовать.

С точки зрения количества элементов, содержащий объект может иметь столько частей, сколько мы хотим. Однако **все части должны иметь ровно один контейнер**.

<https://javarush.com/groups/posts/1967-otnoshenija-mezhdu-klassami-nasledovanie-kompozicija-i-agregirovanie->

<https://habr.com/ru/hub/dev_management/>

<https://habr.com/ru/users/Tomcat/> !!!!!

<https://topjava.ru/blog/oops-concepts-in-java>

<https://topjava.ru/startjava>

<https://topjava.ru/topjava>

<https://github.com/AnastasiyaSachkova/IOS-InterviewQuestions/blob/master/Main/6_oop.md>

**3.2 Структура и выполнение программы**

Основным строительным блоком программы на языке Java являются инструкции (statement). Каждая инструкция выполняет некоторое действие, например, вызовы методов, объявление переменных и присвоение им значений. После завершения инструкции в Java ставится точка с запятой (;). Данный знак указывает компилятору на конец инструкции. Например,

**System.out.println("Hello Java!");**

Данная строка представляет вызов метода System.out.println, который выводит на консоль строку "Hello Java!". В данном случае вызов метода является инструкцией и поэтому завершается точкой с запятой.

Кроме отдельных инструкций распространенной конструкцией **является блок кода**. Блок кода содержит набор инструкций, он заключается в фигурные скобки, а инструкции помещаются между открывающей и закрывающей фигурными скобками:

<https://coderlessons.com/tutorials/java-tekhnologii/izuchite-intellidzh-ideia/intellij-idea-sozdanie-pervogo-java-proekta>

**{**

**System.out.println("Hello!");**

**System.out.println("Welcome to Java!");**

**}**

Вся программу представляется набором взаимодействующих между собой классов и объектов. В следующей программе основу нашей составляет класс Program.

**public class Program{**

**public static void main (String args[]){**

**System.out.println("Hello Java!");**

**}**

**}**

При определении класса вначале идет модификатор доступа public, который указывает, что данный класс будет **доступен всем**, то есть мы сможем его запустить из командной строки. Далее идет ключевое слово class, а затем название класса. После названия класса идет блок кода, в котором расположено содержимое класса.

**Входной точкой в программу на языке Java является метод main**, который определен в классе Program. Именно с него начинается выполнение программы. Он обязательно должен присутствовать в программе. При этом его заголовок может быть только таким:

**public static void main (String args[])**

При запуске приложения виртуальная машина Java ищет в **главном классе** программы метод main с подобным заголовком, и после его обнаружения запускает его.

Вначале заголовка метода идет модификатор public, который указывает, что метод будет доступен извне. Слово static указывает, что метод main - статический, а слово void - что он не возвращает никакого значения. Далее в скобках у нас идут параметры метода - String args[] - это массив args, который хранит значения типа String, то есть строки. При запуске программы через этот массив мы можем передать в программу различные данные. После заголовка метода идет его блок, который содержит набор выполняемых инструкций.

Код программы может содержать комментарии. Комментарии позволяют понять смысл программы, что делают те или иные ее части. При компиляции комментарии игнорируются и не оказывают никакого влияния на работу приложения и на его размер.

В Java есть два типа комментариев: однострочный и многострочный. Однострочный комментарий размещается на одной строке после двойного слеша **//**. А многострочный комментарий заключается между символами **/\*** текст комментария **\*/**. Он может размещаться на нескольких строках.

**3.3 Переменные и константы**

Для хранения данных в программе предназначены переменные. Переменная представляет именованную область памяти, которая хранит значение определенного типа. Каждая переменная имеет тип, имя и значение. Тип определяет, какую информацию может хранить переменная или диапазон допустимых значений.

Переменные объявляются следующим образом:

**тип\_данных имя\_переменной;**

**int x;**

В этом выражении объявляется переменная x типа int. То есть x будет хранить некоторое число не больше 4 байт**.**

В качестве имени переменной может выступать любое произвольное название, которое удовлетворяет следующим требованиям:

* имя может содержать любые алфавитно-цифровые символы, а также знак подчеркивания, при этом первый символ в имени не должен быть цифрой;
* в имени не должно быть знаков пунктуации и пробелов;
* имя не может быть ключевым словом языка Java.

Кроме того, при объявлении и последующем использовании надо учитывать, что Java - регистрозависимый язык, поэтому следующие объявления **int num**; и **int NUM**; будут представлять две разных переменных.

Объявив переменную, мы можем присвоить ей значение:

**int x; // объявление переменной**

**x = 10; // присвоение значения**

**// int x = 10; // объявление и инициализация переменной**

**int x, y; // возможный вариант**

**int x = 8, y = 15; // возможный вариант**

**System.out.println(x);**

**System.out.println(y);**

Отличительной особенностью переменных является то, что мы можем в процессе работы программы изменять их значение.

Начиная с Java 10 в язык было добавлено ключевое слово var, которое также позволяет определять переменную:

**var x = 10;**

**System.out.println(x); // 10**

Слово var ставится вместо типа данных, а сам тип переменной выводится из того значения, которое ей присваивается. Например, переменной x присваивается число 10, значит, переменная будет представлять тип int.

Но если переменная объявляется с помощью var, то мы обязательно должны инициализировать ее, то есть предоставить ей начальное значение, иначе мы получим ошибку, как, например, в следующем случае:

**var x; // ! Ошибка, переменная не инициализирована**

**x = 10;**

Кроме переменных, 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**:

final int LIMIT = 5;

System.out.println(LIMIT); // 5

// LIMIT=57; ***// так мы уже не можем написать, так как LIMIT – константа***

**3.4 Какие типы данных есть в Java?**

В Java типы данных делят на **две большие группы: примитивные и ссылочные**.

В состав примитивных типов (или просто примитивов) входят **четыре подвида** и **восемь типов данных**:

1) целые числа (byte, short, int, long);

2) числа с плавающей точкой (float, double);

3) логический (boolean);

4) символьный (char).

**Ссылочные типы данных ещё называют ссылками**. К ним относятся все **классы**, **интерфейсы**, **массивы**, а также **тип данных String**.

Хотя у примитивов и ссылок много общего, между ними есть существенные различия. И главное различие — в том, что именно в них хранится.

**Таблица 3.1 Сравнение свойств примитивных и ссылочных данных**

|  |  |
| --- | --- |
| Примитивные переменные | Ссылочные переменные |
| Хранят **значение** | Хранят **адрес объекта в памяти**, на который ссылаются (отсюда и название). Используются для **доступа** к объектам (его нельзя получить, если на объект нет ссылки) |
|  |  |
| Создаются **присваиванием** значения | Создаются через **конструкторы** классов (присваивание только создаёт вторую **ссылку** на существующий объект) |
| Имеют строго заданный **диапазон** допустимых значений | По умолчанию их значение — null |
| В аргументы методов попадают **копии** значения переменной (это передача по значению) | В методы передаётся **значение ссылки** — операция выполняется над оригинальным объектом, на который ссылается переменная |
|  | Могут использоваться для ссылки на любой объект объявленного или совместимого типа |

Вот пример использования примитивных и ссылочных типов данных:

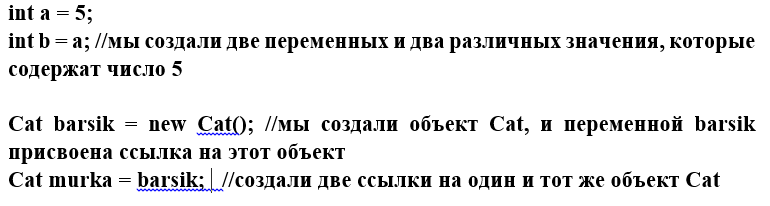


Рисунок 3.1

У примитивов есть строгие рамки допустимых значений по умолчанию и диапазоны значений — для удобства они собраны их в таблицу (рисунок 3.2).

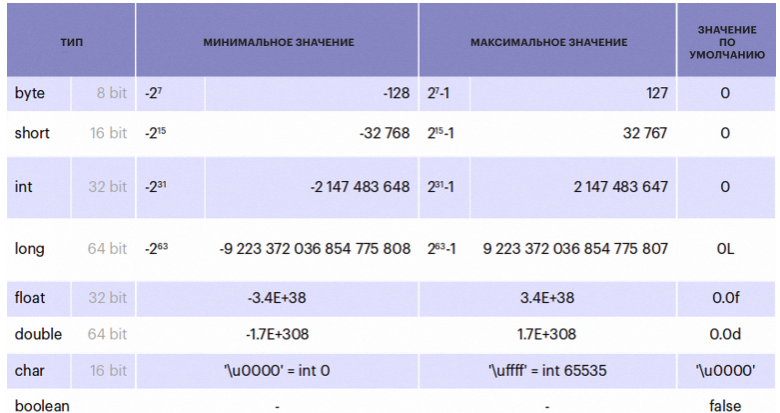


Рисунок 3.2 Диапазоны значений примитивов в языке Java.

У каждой переменной есть значение по умолчанию. Оно присваивается при её создании.

Значение по умолчанию для примитивных числовых типов равно 0 (0 для целочисленных, 0.0d и 0.0f для **double** и **float** соответственно), для **boolean** — **false**, а для типа **char** — \u0000.

На рисунке 3.3 это показано более наглядно!

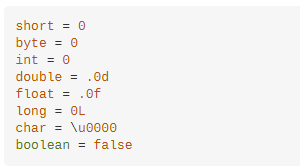


Рисунок 3.3. Значения по умолчанию примитивных типов в Java[[1]](#footnote-1).

Итак, рассмотрим систему встроенных базовых типов данных, которая используется для создания переменных в Java. А она представлена следующими типами.

Java имеет 8 *примитивных типов данных*, а именно:

boolean, byte, short, char, int, long, float и double.

Все остальные типы являются *ссылочными* типами, включая все типы массивов и встроенные типы объектов / классы, которые имеют особое значение на языке Java, например, String, Class и Throwable и его подклассы.

Итак, рассмотрим систему встроенных базовых типов данных, которая используется для создания переменных в Java. А она представлена следующими типами.

**boolean: хранит значение true или false**

boolean isActive = false;

boolean isAlive = true;

**byte: хранит целое число от -128 до 127 и занимает 1 байт**

byte a = 3;

byte b = 8;

**short: хранит целое число от -32768 до 32767 и занимает 2 байта**

short a = 3;

short b = 8;

**int: хранит целое число от -2147483648 до 2147483647 и занимает 4 байта**

int a = 4;

int b = 9;

**long: хранит целое число от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 и занимает 8 байт**

long a = 5;

long b = 10;

**float: хранит число с плавающей точкой от -3.4\*1038 до 3.4\*1038 и занимает 4 байта**

**double: хранит число с плавающей точкой от ±4.9\*10-324 до ±1.7976931348623157\*10308 и занимает 8 байт**

double x = 8.5;

double y = 2.7;

**char: хранит одиночный символ в кодировке UTF-16 и занимает 2 байта, поэтому диапазон хранимых значений от 0 до 65535**

При этом переменная может принимать только те значения, которые соответствуют ее типу. **Если переменная представляет целочисленный тип, то она не может хранить дробные числа**.

**3.5 Особенности работы с типами данных в Java**

**Целые числа**

Все целочисленные литералы, например, числа 10, 4, -5**, воспринимаются как значения типа int**.

Однако мы можем присваивать **целочисленные литералы** другим целочисленным типам: byte, long, short. В этом случае Java автоматически осуществляет соответствующие преобразования:

byte a = 1;

short b = 2;

long c = 2121;

Однако, в следующей строке будет генерироваться ошибка:

long num = 2147483649;

Здесь число 2147483649 является **допустимым для типа long**, но **выходит за предельные значения для типа int**. И так как **все целочисленные значения по умолчанию расцениваются как значения типа int**, то компилятор укажет нам на ошибку. Чтобы решить проблему, надо добавить к числу суффикс l или L, который указывает, что число представляет тип long: **long num = 2147483649L**

Как правило, значения для целочисленных переменных задаются в десятичной системе счисления, однако мы можем применять и другие системы счисления. Например,

**int num111 = *0x*6F; // 16-теричная система, число 111**

**int num8 = *0*10; // 8-ричная система, число 8**

**int num13 = *0b*1101; // 2-ичная система, число 13**

Для задания шестнадцатеричного значения после символов ***0x*** указывается число в шестнадцатеричном формате. Таким же образом восьмеричное значение указывается после символа ***0***, а двоичное значение - после символов ***0b***.

**Также целые числа поддерживают разделение разрядов числа с помощью знака подчеркивания**:

**int x = 123\_456;**

**int y = 234\_567\_\_789;**

**System.out.println(x); // 123456**

**System.out.println(y); // 234567789**

**Числа с плавающей точкой**

При присвоении переменной типа float дробного литерала с плавающей точкой, например, 3.1, 4.5 и т.д., Java **автоматически рассматривает этот литерал как значение типа double**. И чтобы указать, что данное значение должно рассматриваться как float, нам надо использовать суффикс ***f***:

**float fl = 30.6f;**

**double db = 30.**6;

Хотя в данном случае обе переменных имеют практически одно значения, но эти значения будут по-разному рассматриваться и будут занимать разное место в памяти.

**Символы и строки**

В качестве значения **переменная символьного типа получает одиночный символ, заключенный в одинарные кавычки**: **char ch='e'**;.

**Кроме того, переменной символьного типа также можно присвоить целочисленное значение от 0 до 65535**. В этом случае переменная опять же будет хранить символ, а целочисленное значение будет указывать на номер символа в таблице символов Unicode (UTF-16). Например,

**char ch=102; // символ 'f'**

**System.out.println(ch);**

Еще одной формой задания **символьных переменных** является шестнадцатеричная форма: переменная получает значение в шестнадцатеричной форме, которое следует после символов "\u". Например, char ch='\u0066'; опять же будет хранить символ 'f'.

**Символьные переменные не стоит путать со строковыми**, 'a' не идентично "a".

**Строковые переменные представляют объект String**, который в отличие от char или int **не является примитивным типом** в Java:

**String hello = "Hello...";**

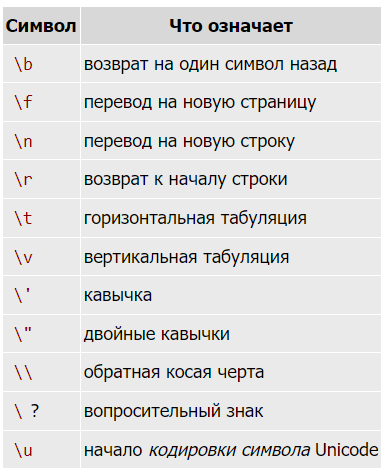
**System.out.println(hello);**

**Управляющие последовательности (**[**https://intuit.ru/studies/courses/569/425/lecture/9665?page=3**](https://intuit.ru/studies/courses/569/425/lecture/9665?page=3)

**String text = "Hello \nworld";**

**System.out.println(text)**

Таблица 3.1 Управляющие символы



**Управляющие последовательности – символы Unicode**

*Управляющая последовательность* может содержать несколько символов. Например, символы национальных алфавитов могут кодироваться последовательностью "\u", после которой идет код символа в шестнадцатеричной кодировке для кодовых таблиц *UTF-16* или UTF-8.

Например:

* \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

Начиная с версии 15 Java поддерживает тестовые блоки (text blocks) - многострочный текст, облеченный в тройные кавычки. Рассмотрим, в чем их практическая польза. Например, выведем большой многострочный текст:

**String text = "Вот мысль, которой весь я предан,\n"+**

**"Итог всего, что ум скопил.\n"+**

**"Лишь тот, кем бой за жизнь изведан,\n"+**

**"Жизнь и свободу заслужил.";**

**System.out.println(text);**

С помощью операции **+** мы можем присоединить к одному тексту другой, причем продолжение текста может располагаться на следующей строке. Чтобы при выводе текста происходил перенос на следующую строку, применяется последовательность **\n**.

**Текстовые блоки, которые появились в JDK15, позволяют упростить написание многострочного текста**:

**String text = """**

**Вот мысль, которой весь я предан,**

**Итог всего, что ум скопил.**

**Лишь тот, кем бой за жизнь изведан,**

**Жизнь и свободу заслужил.**

**""";**

**System.out.println(text);**

Весь текстовый блок оборачивается в тройные кавычки, при этом не надо использовать соединение строк или последовательность \n для их переноса. Результат выполнения программы будет тем же, что и в примере выше.

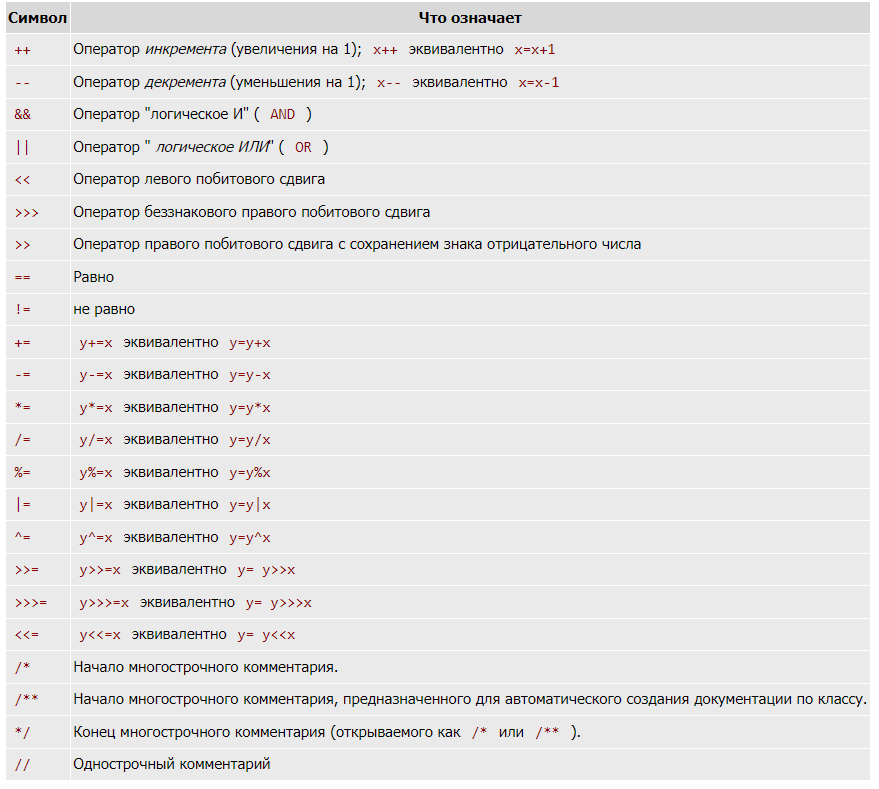
**3.6 Специальные символы**

**Простые специальные символы**

Таблица 3.2

|  |
| --- |
|  |

**Составные специальные символы**



**3.7 Зарезервированные слова языка Java**

Это слова, зарезервированные для синтаксических конструкций языка, причем их назначение нельзя переопределять внутри программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Таблица 1.2**. | | | | |
| abstract | boolean | break | byte | case |
| catch | char | class | const | continue |
| default | do | double | else | enum |
| extends | false | final | finally | float |
| for | Goto | if | implements | import |
| Instanceo f | int | interface | long | native |
| new | null | package | private | protected |
| public | return | short | static | super |
| switch | synchronized | this | throw | throws |
| transient | true | try | void | volatile |
| while |  |  |  |  |

**3.8 Типичные ошибки**

* Путают языки Java и JavaScript, либо считают, что JavaScript – это интерпретируемый вариант Java. Хотя эти языки не имеют друг к другу никакого отношения.
* Ошибочно считают, что приложение Java может быть запущено на любом компьютере без установки исполняющей среды (*JRE*).
* Не различают приложения (applications) и апплеты (applets).
* При записи шестнадцатеричного числа вида 0x… вместо ведущего нуля пишут букву O.
* Ошибочно считают, что в идентификаторах Java нельзя использовать символы национальных алфавитов.
* Ошибочно считают, что не имеет значения, в каком регистре набраны символы идентификатора (характерно для тех, кто раньше программировал на PASCAL или FORTRAN).

<https://intuit.ru/studies/courses/569/425/lecture/9665?page=4>

**Задание**

1. Рассмотреть, понять, реализовать простые примеры классы – оболочки Java.
   1. <https://translated.turbopages.org/proxy_u/en-ru.ru.b287df6d-63e272ef-7a05abcb-74722d776562/https/www.baeldung.com/java-wrapper-classes>
   2. <https://www.examclouds.com/ru/java/java-core-russian/obolochki-tipov>
   3. <https://javadevblog.com/klassy-obolochki-v-java-rukovodstvo-s-primerami.html>
   4. <https://topjava.ru/blog/primitives-and-referenses>
2. Разобраться в деталях и возможностях IntelliJ Idea Community
3. Решить задачи
   1. Напишите программу на Java, которая напечатает «Hello» на экране, а затем напечатайте свое имя в отдельной строке.
   2. Напишите программу на Java для вывода суммы двух чисел. Перейти к редактору
   3. Тестовые данные: 74 + 36
   4. В переменной n хранится натуральное трёхзначное число. Создайте программу, вычисляющую и выводящую на экран сумму цифр числа n.
   5. В переменных q и w хранятся два натуральных числа. Создайте программу, выводящую на экран результат деления q на w с остатком.

<https://learntutorials.net/ru/java/topic/148/примитивные-типы-данных> !!!!

<https://python-scripts.com/sleep>

1. <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html> - Документация Oracle [↑](#footnote-ref-1)