Оглавление

[Примитивные типы данных](#_Toc55136714)

[Логический тип boolean](#_Toc55136715)

[Целочисленные типы](#_Toc55136716)

[Вещественные типы](#_Toc55136717)

[Символьный тип char](#_Toc55136718)

[Операции и операторы над примитивными типами](#_Toc55136719)

[Операторы сравнения](#_Toc55136720)

[Логические операторы (boolean)](#_Toc55136721)

[Арифметические операции](#_Toc55136722)

[Побитовые операции](#_Toc55136723)

[Логические операции](#_Toc55136724)

[Побитовые сдвиги](#_Toc55136725)

[Полезные классы для примитивов](#_Toc55136726)

[Ссылочные типы данных](#_Toc55136727)

[Классы-обертки, автоупаковка и автораспаковка](#_Toc55136728)

[Приведение типов](#_Toc55136729)

[Пул интов](#_Toc55136730)

[Пул строк](#_Toc55136731)

[String](#_Toc55136732)

[Особенности String](#_Toc55136733)

[Почему String неизменяемый и финализированный класс?](#_Toc55136734)

[intern()](#_Toc55136735)

[Можно ли использовать строки в конструкции switch?](#_Toc55136736)

[Почему не рекомендуется изменять строки в цикле? Что рекомендуется использовать?](#_Toc55136737)

[Почему char[] предпочтительнее String для хранения пароля?](#_Toc55136738)

[Почему строка является популярным ключом в HashMap в Java?](#_Toc55136739)

[String, StringBuffer и StringBuilder](#_Toc55136740)

[Многомерные массивы](#_Toc55136741)

[Сигнатура метода](#_Toc55136742)

[main()](#_Toc55136743)

[Каким образом передаются переменные в методы, по значению или по ссылке?](#_Toc55136744)

# Примитивные типы данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| логический тип | boolean | false | 4/1 | true/false |
| целочисленные типы | byte | 0 | 1 | -128 … 127 |
| short | 0 | 2 | -32 768 … 32 767 |
| int | 0 | 4 | ±2 ∙ 109 |
| long | 0L | 8 | ±9 ∙ 1018L |
| вещественные типы | float | 0.0f | 4 | ±3.4 ∙ 1038f |
| double | 0.0d | 8 | ±1.7 ∙ 10308d |
| символьный тип | char | '\u0000' | 2 | 0 …  65 535 |

\* размер в байтах

Свойства:  
∙ имена примитивов — ключевые слова языка;  
∙ переменная примитивного типа представляет собой ячейку памяти, в которой хранится значение переменной;  
∙ примитивные типы передаются по значению (созданием копией);  
∙ не может быть пустой и при инициализации будет иметь дефолтное значение, указанное в таблице.

## Логический тип boolean

Значения: true/false.

Значение по умолчанию: false.

Размер: 4 байта и 1 байт в массиве.

Не конвертируется в другие примитивны типы и наоборот.

Применяются логические операторы и являются результатом операторов сравнения.

## Целочисленные типы

|  |  |  |  |
| --- | --- | --- | --- |
| byte | 0 | 1 байт | -128  127 |
| short | 0 | 2 байта | -32 768  32 767 |
| int | 0 | 4 байт | -2 147 483 648  2 147 483 647  (±2 ∙ 109) |
| long | 0L | 8 байт | -9 223 372 036 854 775 808L  9 223 372 036 854 775 807L  (±9 ∙ 1018) |

Целочисленные литералы имеют по умолчанию тип int.

Запись в разных системах счисления:

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11. | int decimal = 99;  // 10  int octal = 0755;  // 8  int hex = 0xFF;  // 16  int binary = 0b101;  // 2 |

Некоторые особенности:

|  |  |
| --- | --- |
| 1.  2.  3. | int tenMillion = 10\_000\_000;  long tenBillion = 10\_000\_000\_000L; |

Применяются операторы сравнения, арифметические операции, побитовые операции.

Особые случаи:

∙ деление на 0 — исключение (java.lang.ArithmeticException: / by zero);

∙ переполнение переменной ошибкой не считается.

|  |  |
| --- | --- |
| 1.  2.  3.  4. | byte b = 127;  b++;  // -128 |

## Вещественные типы

|  |  |  |  |
| --- | --- | --- | --- |
| float | 0.0f | 4 байта | ±3.4 ∙ 1038f |
| double | 0.0d | 8 байт | ±1.7 ∙ 10308d |

Количество бит (m — мантисса, e — экспонента):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| тип | всего | ± | m | e |
| float | 32 | 1 | 23 | 8 |
| double | 64 | 1 | 52 | 11 |

±m ∙ 2e

Стандарт IEEE 754 (детали в представлении вещественных чисел).

Вещественные литералы имеют по умолчанию тип double.

Запись числа:

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12. | double simple = -1.234;  double exponential = -123.4e-2;  // 123.4 \* 10^2, экспоненциальный  double hex = 0x1.Fp10;  // 1.F \* 2^10  // мантисса в 16-ричной системе счисления  float floatWithSuffix = 36.6f;  double doubleWithSuffix = 4d; |

Применяются операторы сравнения, арифметические операции.

Особые случаи:

∙ существует понятие бесконечность (Infinity) со знаком (результат деления числа на ноль) (можно совершать арифметически операции).

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5. | double positiveInfinity = 1.0 / 0.0;  // Infinit  double negativeInfinify = -1.0 / 0.0;  // -Infinity |

∙ специальное значение NaN (деление нуля на ноль, сложение плюс и минус бесконечностей, умножение нуля на бесконечность) — Not-a-Number (любые арифметические операции — NaN, не равен самому себе).

|  |  |
| --- | --- |
| 1.  2. | double nan = 0.0 / 0.0;  // NaN |

∙ ограниченная точность, поэтому вещественные числа сравнивают не оператором ==, а проверяют |a – b| < ε.

## Символьный тип char

Размер: 2 байта.

Значение по умолчанию: '\u0000'.

Диапазон значений: 0 … 65 535 (216) (беззнаковый).

Хранит в себе символ кодировки Unicode (UTF-16). Можно работать как с целым числом (операторы сравнения, арифметические и побитовые логические операции).

Символьные литералы записываются в программе в одиночных кавычках (в отличии от строк). Внутри может находится ровно один символ либо явно 'a', либо в виде специальной escape‑последовательности: '\t' (табуляция), '\n' (перевод строки), '\r' (возврат каретки), '\'' (одиночная кавычка), '\\' (обратный слеш). Также есть универсальная escape‑последовательность для ввода любого символа Unicode по его коду '\u03A9'.

Источники:

∙ Курс на степике «Java. Базовый курс»: <https://stepik.org/course/187/syllabus>

# Операции и операторы над примитивными типами

## Операторы сравнения

|  |  |
| --- | --- |
| < | меньше |
| <= | меньше или равно |
| > | больше |
| >= | больше или равно |
| == | равно |
| != | не равно |

Результат сравнения — boolean. Сравнивать можно целочисленные (char) и вещественные типы.

## Логические операторы (boolean)

|  |  |
| --- | --- |
| ! | отрицание |
| & | И, конъюнкция |
| &= |
| && | сокращенная И |
| | | ИЛИ, дизъюнкция |
| |= |
| || | сокращенная ИЛИ |
| ^ | исключающая ИЛИ |
| ^= |

При сокращенных И и ИЛИ правый операнд вычисляется при необходимости.

## Арифметические операции

|  |  |
| --- | --- |
| + | сложение |
| += |
| - | вычитание |
| -+ |
| \* | умножение |
| \*= |
| / | деление |
| /= |
| % | остаток от деления |
| %= |
| ++ | инкремент |
| -- | декремент |
| ++a | сначала a + 1 |
| a++ | сначала вернет значение |

Целочисленные (char) и вещественные типы.

## Побитовые операции

Применяются только к целым числам (char).

### Логические операции

|  |  |
| --- | --- |
| ~ | отрицание |
| & | И |
| &= |
| | | ИЛИ |
| |= |
| ^ | исключающее ИЛИ |
| ^= |

### Побитовые сдвиги

|  |  |
| --- | --- |
| >> | арифметический сдвиг вправо |
| >>= |
| >>> | логический сдвиг вправо |
| >>>= |
| << | сдвиг влево |
| <<= |

Арифметический сдвиг вправо — младшие биты теряются, освободившие старшие позиции заполняются знаковым битом, логический сдвиг вправо — освободившиеся позиции заполняются нулями, сдвиг влево — теряются старшие биты, а младшие позиции заполняются нулями. Правым операндов операций логических сдвигов может быть любое число, но Java возьмет остаток от деления этого число на размер целочисленного типа. Например, для int будет вычислен остаток от деления на 32. Фактический сдвиг значения переменной a будет выполнен на получившееся количество бит.

Источники:

∙ Курс на степике «Java. Базовый курс»: <https://stepik.org/course/187/syllabus>

# Полезные классы для примитивов

java.lang.Math — считать логарифмы, квадратные корни, тригонометрические функции, округлять дробные числа, различные константы, методы min()/max() и многое другое.

java.math.BigInteger — работа с целочисленными типами, если не влезает в long.

java.math.BigDecimal — работа с вещественными типами без потери точности.

Источники:

∙ Курс на степике «Java. Базовый курс»: <https://stepik.org/course/187/syllabus>

# Ссылочные типы данных

Типы данных, не указанные выше, являются ссылочными.

Отличия от свойств примитивов:  
∙ имена ссылочных типов не являются ключевыми словами языка, поэтому можно свободно создавать и использовать собственные ссылочные типы, а также называть методы или переменные (противоречит наименованию классов и переменных/методов);  
∙ переменная ссылочного типа представляет собой ячейку памяти, содержащую ссылку на участок памяти, представляющий собой объект;  
∙ при присваивании значения одной переменной в другую или при передаче ее в качестве параметра в метод выполняется копирование ссылки; получаем вторую ссылку, указывающую на тот же объект; если через 2‑ую ссылку объект модифицируется, то те же самые изменения будут видны и через первую ссылку, потому что это тот же самый объект (строки являются исключения — string pool);  
∙ может быть пустой, т. е. не указывать ни на какой объект. Для этого есть специальное значение null, которое можно присвоить любой переменной ссылочного типа.

Любая переменная ссылочного типа либо пустая null, либо указывает на существующий объект.

Источники:

∙ Курс на степике «Java. Базовый курс»: <https://stepik.org/course/187/syllabus>

# Классы-обертки, автоупаковка и автораспаковка

У каждого примитива есть соответствующая класс-обертка: Byte, Short, Integer, Long, Float, Double, Character, Boolean. Классы-обертки занимают больше места, но в отличии от примитивов могут применять к значениям какие‑либо методы или хранить значение null.

Автоупаковка и автораспаковка — преобразования примитивных типов в соответствующие объекты классов-оберток и наоборот без явного использования конструктора класса, т. е.  при помощи оператора =, либо при передаче в параметры метода.

Автоупаковка требует точного соответствия типов, иначе ошибка клмпиляции.

Автоупаковке в классы-обертки могут быть подвергнуты как переменные примитивных типов, так и константы времени компиляции (литералы и final-примитивы). При этом литералы должны быть синтаксически корректными для инициализации переменной исходного примитивного типа.

Автоупаковка констант примитивных типов допускает более широкие границы соответствия. В этом случае компилятор способен предварительно осуществлять неявное расширение/сужение типа примитивов:

неявное расширение/сужение исходного типа примитива до типа примитива соответствующего классу-обертке (для преобразования int в Byte, сначала компилятор самостоятельно неявно сужает int к byte)

автоупаковку примитива в соответствующий класс-обертку. Однако, в этом случае существуют два дополнительных ограничения: a) присвоение примитива обертке может производится только оператором = (нельзя передать такой примитив в параметры метода без явного приведения типов) b) тип левого операнда не должен быть старше чем Character, тип правого не дожен старше, чем int: допустимо расширение/сужение byte в/из short, byte в/из char, short в/из char и только сужение byte из int, short из int, char из int. Все остальные варианты требуют явного приведения типов).

Дополнительной особенностью целочисленных классов-оберток, созданных автоупаковкой констант в диапазоне -128 ... +127 я вляется то, что они кэшируются JVM. Поэтому такие обертки с одинаковыми значениями будут являться ссылками на один объект.

# Приведение типов

Неявное преобразование типов — безопасное преобразование, в результате которых не может произойти потеря данных.

Старшие разряды более широкого типа заполняются знаковым битом исходного значения: byte -> short, short -> int, int -> long.

Старшие биты заполняются нулями: char -> int/long.

Целочисленные типы автоматически конвертируются в вещественные. При этом возможна потеря точности (в некоторых младших разрядах; неизменным будет только порядок и старшие разряды, количество которых определяется емкостью вещественного типа).

При явное преобразование возможна потеря информации, поэтому компилятор требует явного подтверждения вашего намерения (оператор приведения типа).

При приведении более емкого целого типа к менее емкому целому лишние старшие биты просто отбрасываются:

|  |  |
| --- | --- |
| 1.  2.  3.  4. | int intValue = 1024;  byte byteValue = (byte) intValue;  // 0 |

При приведении типа с плавающей точкой к целому числу дробная часть просто отбрасывается (математическое округление не происходит):

|  |  |
| --- | --- |
| 1.  2.  3.  4. | double pi = 3.14;  int intFromDouble = (int) pi;  // 3 |

Слишком большое по модулю вещественное число при приведении к целому превращается в максимальное по модулю представимое целое число того же знака:

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5. | float largeFloat = 1e20f;  // 2 \* 10^20  int intFromLargeFloat = (int) largeFloat;  // 2 \* 10^9 |

Слишком большой double при приведении к float превращается в бесконечность:

|  |  |
| --- | --- |
| 1.  2.  3.  4. | double largeDouble = 1e100;  float floatFromLargeDouble = (float) largeDouble;  // Infinity |

Автоматическое расширение — принудительное приведение типа (при применении бинарного, арифметического или побитового операторов к 2‑м операндам, операнды приводятся к одному типу).

Общий тип выбирается по следующим правилам. Если один из операндов имеет тип double, то оба приводятся к double, иначе если один из операндов имеет тип float, то оба приводятся к float, иначе если один из операндов имеет тип long, то оба приводятся к long и иначе оба приводятся к типу int. Обратите внимание на это последнее правило. Из него следует, когда вы, например, складываете 2 значения типа byte, то получаете int. И присвоить результат обратно в переменную типа байт без явного приведения уже не получится. Зато, когда вы получите от компилятора такую ошибку, вы лишний раз подумаете о возможном переполнении.

Java является строго типизированным языком программирования, а это означает то, что каждое выражение и каждая переменная имеет строго определенный тип уже на момент компиляции. Однако определен механизм приведения типов — способ преобразования значения переменной одного типа в значение другого типа.

В Java существуют несколько разновидностей приведения:

∙ Тождественное. Преобразование выражения любого типа к точно такому же типу всегда допустимо и происходит автоматически.

∙ Расширение примитивного типа. Означает, что осуществляется переход от менее емкого типа к более емкому. Например, от типа byte (длина 1 байт) к типу int (длина 4 байта). Такие преобразование безопасны в том смысле, что новый тип всегда гарантировано вмещает в себя все данные, которые хранились в старом типе и т. о. не происходит потери данных. Этот тип приведения всегда допустим и происходит автоматически.

∙ Сужение примитивного типа. Означает, что переход осуществляется от более емкого типа к менее емкому. При таком преобразовании есть риск потерять данные. Например, если число типа int было больше 127, то при приведении его к byte значения битов старше восьмого будут потеряны. В Java такое преобразование должно совершаться явным образом, при этом все старшие биты, не умещающиеся в новом типе, просто отбрасываются — никакого округления или других действий для получения более корректного результата не производится.

∙ Расширение объектного типа. Означает неявное восходящее приведение типов или переход от более конкретного типа к менее конкретному, т. е. переход от потомка к предку. Разрешено всегда и происходит автоматически.

∙ Сужение объектного типа. Означает нисходящее приведение, т. е. приведение от предка к потомку (подтипу). Возможно только если исходная переменная является подтипом приводимого типа. При несоответствии типов в момент выполнения выбрасывается исключение ClassCastException. Требует явного указания типа.

∙ Преобразование к строке. Любой тип может быть приведен к строке, т. е. к экземпляру класса String.

∙ Запрещенные преобразования. Не все приведения между произвольными типами допустимы. Например, к запрещенным преобразованиям относятся приведения от любого ссылочного типа к примитивному и наоборот (кроме преобразования к строке). Кроме того, невозможно привести друг к другу классы, находящиеся на разных ветвях дерева наследования и т. п.

При приведении ссылочных типов с самим объектом ничего не происходит, — меняется лишь тип ссылки, через которую происходит обращение к объекту.

Для проверки возможности приведения нужно воспользоваться оператором instanceof:

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6. | Parent parent = new Child();  if (parent instanceof Child) {  Child child = (Child) parent;  } |

# Пул интов

В Java есть пул целых чисел в промежутке [-128; 127]. Т. е. если мы создаем Integer в этом промежутке, то вместо того, чтобы каждый раз создавать новый объект, JVM берет их из пула.

# Пул строк

Пул строк — набор строк, хранящийся в Heap.

∙ Пул строк возможен благодаря неизменяемости строк в Java и реализации идеи интернирования строк;

∙ Пул строк помогает экономить память, но по этой же причине создание строки занимает больше времени;

∙ Когда для создания строки используются ", то сначала ищется строка в пуле с таким же значением, если находится, то просто возвращается ссылка, иначе создается новая строка в пуле, а затем возвращается ссылка на нее;

∙ При использовании оператора new создается новый объект String. Затем при помощи метода intern() эту строку можно поместить в пул или же получить из пула ссылку на другой объект String с таким же значением;

∙ Пул строк является примером паттерна «Приспособленец».

# String

## Особенности String

∙ неизменяемый и финализированный тип данных;  
∙ хранится в пуле строк;  
∙ можно получить, используя двойные кавычки;  
∙ можно использовать оператор + для конкатенации строк;  
∙ с Java 7 строки можно использовать в конструкции switch.

## Почему String неизменяемый и финализированный класс?

Преимущества в неизменности строк:

∙ Пул строк возможен только потому, что строка неизменяемая, таким образом виртуальная машина сохраняет больше свободного места в Heap, поскольку разные строковые переменные указывают на одну и ту же переменную в пуле. Если бы строка была изменяемой, то интернирование строк не было бы возможным, потому что изменение значения одной переменной отразилось бы также и на остальных переменных, ссылающихся на эту строку.

∙ Если строка будет изменяемой, тогда это станет серьезной угрозой безопасности приложения. Например, имя пользователя базы данных и пароль передаются строкой для получения соединения с базой данных и в программировании сокетов реквизиты хоста и порта передаются строкой. Т. к. строка неизменяемая, ее значение не может быть изменено, в противном случае злоумышленник может изменить значение ссылки и вызвать проблемы в безопасности приложения.

∙ Неизменяемость позволяет избежать синхронизации: строки безопасны для многопоточности и один экземпляр строки может быть совместно использован различными потоками.

∙ Строки используются classloader и неизменность обеспечивает правильность загрузки класса.

∙ Поскольку строка неизменяемая, ее hashCode() кэшируется в момент создания и нет необходимости рассчитывать его снова. Это делает строку отличным кандидатом для ключа в HashMap, т. к. его обработка происходит быстрее.

## intern()

Метод intern() используется для сохранения строки в пуле строк или получения ссылки, если такая строка уже находится в пуле.

## Можно ли использовать строки в конструкции switch?

Да, начиная с Java 7 в операторе switch можно использовать строки, ранние версии Java не поддерживают этого. При этом:  
∙ участвующие строки чувствительны к регистру;  
∙ используется метод equals() для сравнения полученного значения со значениями case, поэтому во избежание NPE стоит предусмотреть проверку на null;  
∙ согласно документации, Java 7 для строк в switch, компилятор Java формирует более эффективный байткод для строк в конструкции switch, чем для сцепленных условий if-else.

# Почему не рекомендуется изменять строки в цикле? Что рекомендуется использовать?

StringBuilder???

# Почему char[] предпочтительнее String для хранения пароля?

С момента создания строка остается в пуле, до тех пор, пока не будет удалена сборщиком мусора. Поэтому, даже после окончания использования пароля, он некоторое время продолжает оставаться доступным в памяти и способа избежать этого не существует. Это представляет определенный риск для безопасности, поскольку кто‑либо, имеющий доступ к памяти сможет найти пароль в виде текста. В случае использования массива символов для хранения пароля имеется возможность очистить его сразу по окончанию работы с паролем, позволяя избежать риска безопасности, свойственного строке.

# Почему строка является популярным ключом в HashMap в Java?

Поскольку строки неизменяемы, их хэш код вычисляется и кэшируется в момент создания, не требуя повторного пересчета при дальнейшем использовании. Поэтому в качестве ключа HashMap они будут обрабатываться быстрее.

# String, StringBuffer и StringBuilder

При изменении объекта String каждый раз создаются новые объекты, а не изменяется исходная строка. Классы StringBuffer и StringBuilder позволяют изменять исходную строку. Первый используется для многопоточного режима, второй — в обычной среде, т. к. будет выполняться быстрее.

# Многомерные массивы

В Java существуют массив массивов. Это не многомерный массив.

# Сигнатура метода

Имя метода и типы параметров (порядок и количество имеет значение).

# main()

∙ точка входа в программу;  
∙ может быть несколько;  
∙ при отсутствии код скомпилируется, но при запуске выдаст ошибку `Error: Main method not found`.

# Каким образом передаются переменные в методы, по значению или по ссылке?

В Java параметры всегда передаются только по значению, что определяется как «скопировать значение и передать копию». С примитивами это будет копия содержимого. Со ссылками — тоже копия содержимого, т. е. копия ссылки. При этом внутренние члены ссылочных типов через такую копию изменить возможно, а вот саму ссылку, указывающую на экземпляр — нет.