Оглавление

Типы данных

Примитивные типы данных

char

Ссылочные типы данных

Классы-обертки, автоупаковка
и автораспаковка

Приведение типов

Пул интов

Пул строк

String

Особенности String

Почему String неизменяемый
и финализированный класс?

intern()

Можно ли использовать строки
в конструкции switch?

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

Почему char[] предпочтительнее String
для хранения пароля?

Почему строка является популярным ключом
в HashMap в Java?

String, StringBuffer и StringBuilder

Многомерные массивы

Сигнатура метода

main()

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

Типы данных

Примитивные типы данных

Тип	Дефолтное значение	Объем памяти, байт	Диапазон значений
byte	0	1	-128 127
short	0	2	-32 768 32 767
int	0	4	-2 147 483 648 2 147 483 647 $(2 \cdot 10^9)$
long	0L	8	-9223372036854775808L 9223372036854775807L $(9 \cdot 10^{18})$
float	0.0f	4	$1.4 \cdot 10^{-45}$f $3.4 \cdot 10^{38}$f
double	0.0d	8	$4.9 \cdot 10^{-324}$ $1.7 \cdot 10^{-308}$
char	'\u0000'	2	0 65 536
boolean	false	4 и 1 в массиве	true false

char

Тип char хранит в себе символы. В Java для char используется кодировка Unicode. Каждый символ имеет размер 2 байта и диапазон значение 0-65 536. Символы в отличии от строки заключаются в одинарные кавычки. В кавычках можно указать как сам символ, так и его номер (`'\u0000'`).

Может выступать как символ, так и как целое число, к которому можно применять сложение и вычитание.

Ссылочные типы данных

`String` — `null`.

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

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

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.