

Signed and Unsigned: short and char

Компилятор позволяет использовать их взаимозаменяемо в некоторых случаях, как показано здесь:

short bird = 'd';

char mammal = (short)83;

При печати каждой переменной отображается значение, связанное с их типом.

System.out.println(bird); // Prints 100

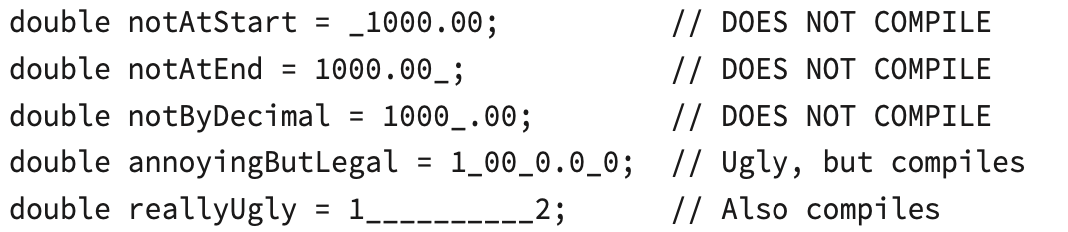
System.out.println(mammal); // Prints S

Однако это использование не без ограничений. Если вы попытаетесь установить значение вне диапазона short или char, компилятор сообщит об ошибке.

short reptile = 65535; // DOES NOT COMPILE

char fish = (short)-1; // DOES NOT COMPILE

Literals and the Underscore Character



Declaring Variables

Идентификаторы должны начинаться с буквы, символа $ или символа \_.

Идентификаторы могут включать числа, но не начинаться с них.

Начиная с Java 9, одиночное подчеркивание \_ не допускается в качестве идентификатора.

Вы не можете использовать то же имя, что и зарезервированное слово Java.

Эти примеры законны:

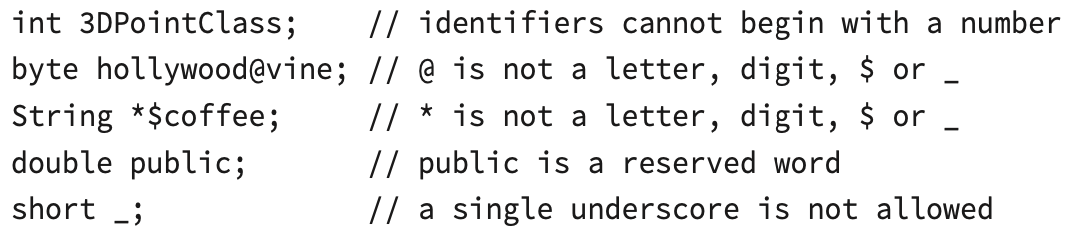
long okidentifier;

float $OK2Identifier;

boolean \_alsoOK1d3ntifi3r;

char \_\_SStillOkbutKnotsonice$;

Эти примеры не являются законными:



Декларация нескольких переменных:

int p1, p2, p3; //по умолчанию 0

void m() {

String s1, s2; //не инициализировались, только объявились

String s3 = "yes", s4 = "no";

int i1, i2, i3 = 0; //инициализировалась только i3

int num, String value; //DOES NOT COMPILE

}

Creating Local Variables

Локальная переменная - это переменная, определенная в конструкторе, методе или блоке инициализатора.

Локальные переменные не имеют значения по умолчанию и должны быть инициализированы перед использованием. Кроме того, компилятор сообщит об ошибке, если вы попытаетесь прочитать неинициализированное значение.

Если инициализация не происходит в одной из ветвей программы, но после происходит попытка использования, код не скомпилируется.

Passing Constructor and Method Parameters

public void checkAnswer() {

boolean value;

findAnswer(value); // DOES NOT COMPILE

}

Introducing var

public void whatTypeAmI() {

var name = "Hello";

var size = 7;

}

var – только локальные пременные. Синтаксическиц сахар. Работает при условии, что компилятор сможет понять, что за тип тут явно подразумевается.

public class VarKeyword {

var tricky = "Hello"; // DOES NOT COMPILE

}

Хотя var не может быть инициализирован нулевым значением без типа, ему может быть присвоено нулевое значение после его объявления, при условии, что базовый тип данных var является объектом.

var n = "myData";

n = null;

var o = (String)null; //если указываешь тип, то можно

var m = 4;

m = null; // DOES NOT COMPILE

public int addition(var a, var b) { // DOES NOT COMPILE

return a + b;

}

var – не зарезервированное слово, так что можно так:

package var;

public class Var {

public void var() {

var var = "var";

}

public void Var() {

Var var = new Var();

}

}

А так нельзя:

public class var { // DOES NOT COMPILE

public var() {

}

}

Общие правила для var:

1. var используется как локальная переменная в конструкторе, методе или блоке инициализатора.

2. var нельзя использовать в параметрах конструктора, параметрах метода, переменных экземпляра или переменных класса.

3. var всегда инициализируется в той же строке (или операторе), где он объявлен.

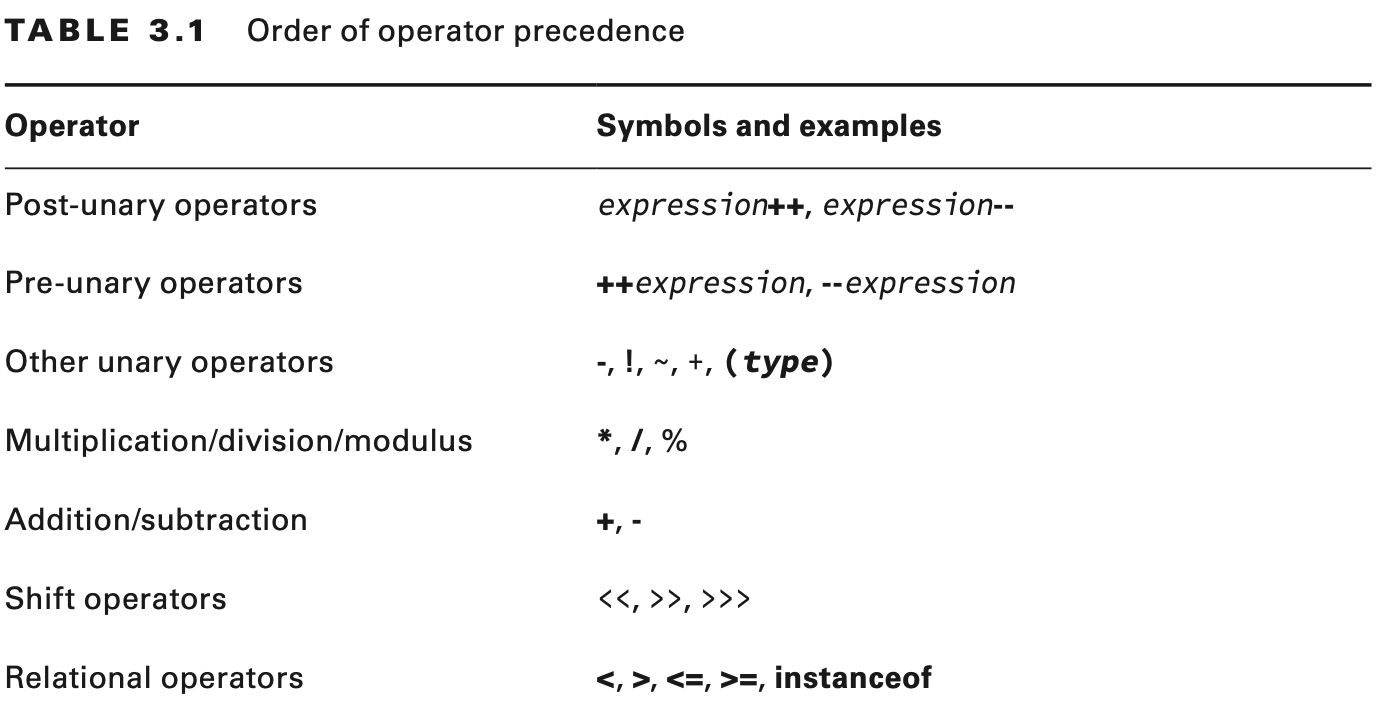
4. Значение переменной может изменяться, а тип - нет.

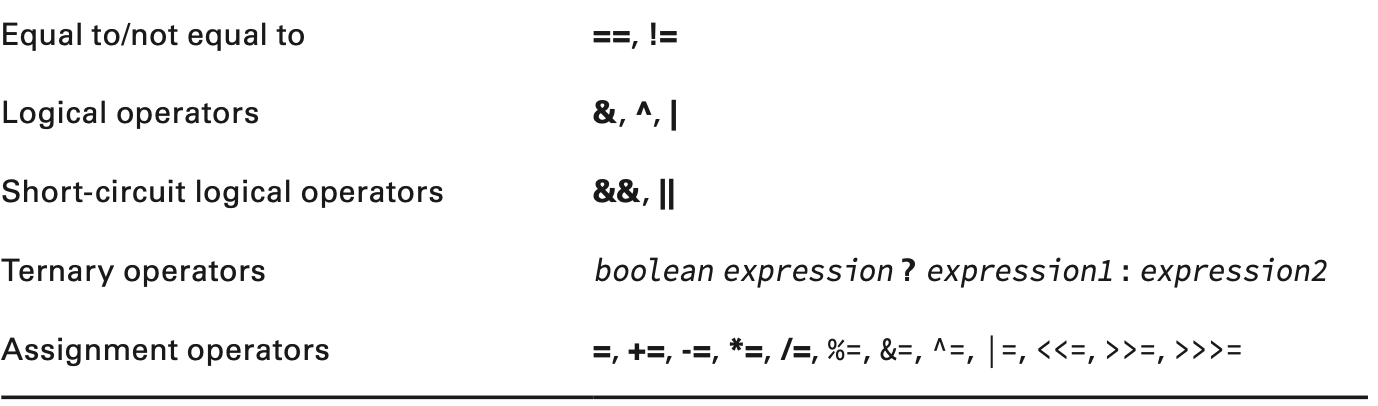
5. var не может быть инициализирован нулевым значением без типа.

6. В объявлении с несколькими переменными нельзя использовать переменную.

7. var - это зарезервированное имя типа, но не зарезервированное слово, что означает, что его можно использовать в качестве идентификатора, кроме имени класса, интерфейса или перечисления.

Операторы





Numeric Promotion

1. Если два значения имеют разные типы данных, Java автоматически повысит одно из значений до большего из двух типов данных.

2. Если одно из значений является целым, а другое - с плавающей запятой, Java автоматически продвинет интегральное значение к типу данных с плавающей запятой.

3. Меньшие типы данных, а именно byte, short и char, сначала повышаются до int каждый раз, когда они используются с двоичным арифметическим оператором Java, даже если ни один из операндов не является int. Унарные операторы исключены из этого правила. Например, применение ++ к short приводит к short.

4. После того, как все продвижение произошло и операнды будут иметь тот же тип данных, результирующее значение будет иметь тот же тип данных, что и его продвинутые операнды.

Casting Values

1. Компилятор автоматически преобразует меньшие типы данных в более крупные.
2. При приведении крупных типов к более мелким явный кастинг необходим.
3. Целочисленные значения присваиваются к переменным с плавающими точкам без ошибок компиляции, хоть и с потерей части значения.

float f = Integer.MAX\_VALUE; // оба по 32 бита

float d = Long.MAX\_VALUE + 10; // в 32 кладем 64

1. При присваивании к целочисленным переменным значений с плавающей точкой необходим явный каст, даже если по размеру они совместимы.

long l = (long) Float.MAX\_VALUE;

1. При касте небольших значений к маленьким типам данных компилятор неявно кастует значения сам, но не может этого делать при присваивании результатов выражения, поэтому тут тоже нужен явный каст.

short mouse = 10;  
short hamster = 3;  
short capybara = mouse \* hamster; // DOES NOT COMPILE

Strings

Стринги иммутабельны. Любой метод изменения стринга создаёт новый стринг.

int length()

String string = "animals";

System.out.println(string.length()); // 7

char charAt(int index)

String string = "animals";

System.out.println(string.charAt(0)); // a

System.out.println(string.charAt(6)); // s

System.out.println(string.charAt(7)); // throws exception

int indexOf(int ch)

int indexOf(int ch, int fromIndex)

int indexOf(String str)

int indexOf(String str, int fromIndex)

String string = "animals";

System.out.println(string.indexOf('a')); // 0

System.out.println(string.indexOf("al")); // 4

System.out.println(string.indexOf('a', 4)); // 4

System.out.println(string.indexOf("al", 5)); // -1

String substring(int beginIndex)

String substring(int beginIndex, int endIndex)

String string = "animals";

System.out.println(string.substring(3)); // mals

System.out.println(string.substring(string.indexOf('m'))); // mals

System.out.println(string.substring(3, 4)); // m

System.out.println(string.substring(3, 7)); // mals

System.out.println(string.substring(3, 3)); // empty string

System.out.println(string.substring(3, 2)); // throws exception System.out.println(string.substring(3, 8)); // throws exception

String toLowerCase()

String toUpperCase()

boolean equals(Object obj)

boolean equalsIgnoreCase(String str)

boolean startsWith(String prefix)

boolean endsWith(String suffix)

String replace(char oldChar, char newChar)

String replace(CharSequence target, CharSequence replacement)

System.out.println("abcabc".replace('a', 'A')); // AbcAbc System.out.println("abcabc".replace("a", "A")); // AbcAbc

boolean contains(CharSequence charSeq)

|  |  |
| --- | --- |
| String trim() | Удаляет пробелы и спец.символы с краёв строки |
| String strip() | То же самое, что trim(), но поддерживает Unicode |
| String stripLeading() | Удаляет пробелы с начала строки |
| String stripTrailing() | Удаляет пробелы с конца строки |

String intern()

Метод intern() возвращает значение из пула строк, если оно есть. В противном случае он добавляет значение в пул строк.

StringBuilder

insert()

StringBuilder sb = new StringBuilder("animals");

sb.insert(7, "-"); // sb = animals-

sb.insert(0, "-"); // sb = -animals-

sb.insert(4, "-"); // sb = -ani-mals-

System.out.println(sb);

StringBuilder delete(int startIndex, int endIndex)

StringBuilder deleteCharAt(int index)

StringBuilder sb = new StringBuilder("abcdef");

sb.delete(1, 3); // sb = adef

sb.deleteCharAt(5); // throws an exception

sb.delete(1, 100); // sb = a, так можно!

StringBuilder replace(int startIndex, int endIndex, String newString)

StringBuilder builder = new StringBuilder("pigeon dirty");

builder.replace(3, 6, "sty");

System.out.println(builder); // pigsty dirty

StringBuilder builder = new StringBuilder("pigeon dirty");

builder.replace(3, 100, "");

System.out.println(builder); // pig

StringBuilder reverse()

StringBuilder sb = new StringBuilder("ABC"); sb.reverse();

The String Pool

Пул строк содержит буквальные значения и константы, которые появляются в вашей программе. Например, «name» является литералом и поэтому попадает в пул строк. myObject.toString () - это строка, но не литерал, поэтому она не попадает в пул строк.

String x = "Hello World";

String y = "Hello World";

System.out.println(x == y); // true

String x = "Hello World";

String z = " Hello World".trim();

System.out.println(x == z); // false

String singleString = "hello world";

String oneLine = "hello " + "world";

String concat = " hello";

concat += "world";

System.out.println(singleString == oneLine); //false

System.out.println(singleString == concat); //false

String x = "Hello World";

String y = new String("Hello World");

System.out.println(x == y); // false

String first = "rat" + 1; // на этапе компиляции выражение вычисляется как rat1 и ложится в пул строк

String second = "r" + "a" + "t" + "1"; // на этапе компиляции выражение вычисляется как rat1 и ложится в пул строк

String third = "r" + "a" + "t" + new String("1");

System.out.println(first == second); // true

System.out.println(first == second.intern()); // true

System.out.println(first == third); // false

System.out.println(first == third.intern()); // true