Шпаргалка к зашите Лабораторной работы №1

10 апреля 2021 г.

1 Язык Java. Особенности языка.

Java — язык программирования общего назначения. Относится к объектно-ориентированным языкам программирования, к языкам с сильной типизацией.

Особенности языка:

- **Простой**

  Синтаксис Java, по существу, представляет собой упрощенный вариант синтаксиса C++. В этом языке **отсутствует потребность в** файлах заголовков, арифметике (и даже в синтаксисе) указателей, структурах, объединениях, перегрузке операций, виртуальных базовых классах и т.п.

  Но создатели Java **не стремились исправить все недостатки языка C++**. Например, синтаксис оператора **switch в Java остался неизменным**.

- **Объектно-ориентированный**

  Особенности Java, связанные с объектами, сравнимы с языком C++. Основное **отличие между ними заключается** в механизме множественного наследования, который в Java заменен более простым понятием интерфейсов.

- **Распределенный**

  Язык Java предоставляет разработчику обшиирную библиотеку программ для передачи данных по протоколу TCP/IP, HTTP и FTP.

  Приложения на Java способны открывать объекты и получать к ним доступ по сети с такой же лёгкостью, как и в локальной файловой системе, используя URL для адресации.

- **Интерпретируемый**

  Интерпретатор Java может выполнять байт-код непосредственно на любой мащине, на которую перенесен интерпретатор. А поскольку процесс компоновки носит в большей степени пошаговый и относительно простой характер, процесс разработки программ может быть заметно ускорен, став более творческим.

- **Надёжный**

  Компилятор Java выявляет такие ошибки, которые в других языках обнаруживаются только на этапе выполнения программы. Кроме того, программисты, потратившие многие часы на поиски ошибки, вызвавшей нарушения данных в памяти из-за неверного указателя, будут обрадованы тем, что в работе с Java подобные осложнения не могут даже в принципе возникнуть.(Java модель указателей исключает воможность записи в произвольно выбранную область памяти и повреждения данных)

- **Безопасный**

  Ниже перечислены некоторые виды нарушения защиты, которые с самого начала нредотвращает система безопасности Java:

  1. **Намеренное переполнение стека выполняемой программы** - один из распространенных способов нарушения защиты, используемых вирусами и "червями".
  2. **Повреждение данных** на участках памяти, находящихся за пределами пространства, выделенного процессу.
  3. **Несанкционированное** чтение файлов и их модификация.

- **Архитектурно-нейтральный**

  Компилятор генерирует объектный файл, формат которого не зависит от архитектуры компьютера.

  Скомпилированная программа может выполняться на любых процессорах, а для ее работы требуется лишь исполняющая система Java. Код, генерируемый компилятором Java, называется **байт-кодом**.

- **Переносимый**

  В отличие от C и C++, ни один из аспектов спецификации Java не зависит от реализации. Разрядность примитивных типов данных и арифметические операции над ними строго определены.

- **Высокоэффективный**

  Обычно интерпретируемый байт-код имеет достаточную производительность, но бывают ситуации, когда требуется еще более высокая производительность.

  Байт-код можно транслировать во время выполнения программы в машинный код для того процессора, на котором выполняется данное приложение.

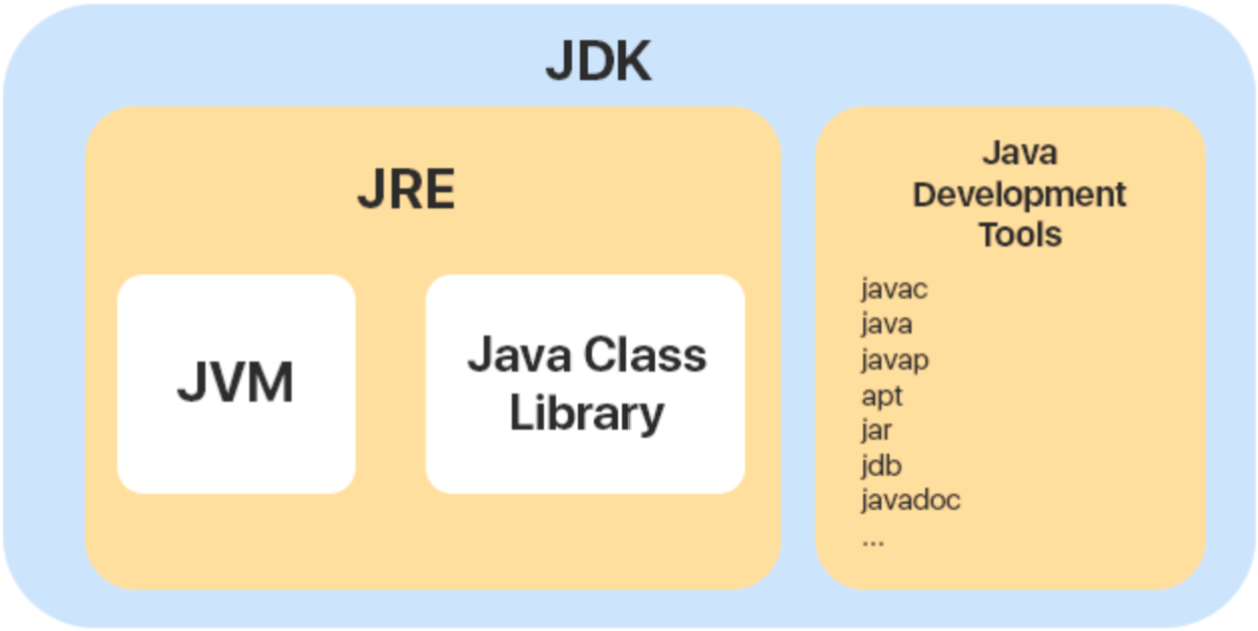
- **Многопотоковый**

  В настоящее время все большее значение приобретает распараллеливание выполняемых задач, поскольку действие закона Мура подходит к концу. В нашем распоряжении теперь имеются не более быстродействующие процессоры, а большее их количество, и поэтому мы должны загрузить их полезной работой, чтобы они не простаивали.

- **Динамический**

  В библиотеки можно свободно включать новые методы и объекты, ни коим образом не затрагивая приложения, пользующиеся библиотеками.

2 Средства разработки. JDK и JRE.



2.1 Java Development Kit

Java Development Kit — комплект разработчика приложений на языке Java. Он включает в себя **Java Development Tools** и среду выполнения Java — **JRE (Java Runtime Environment)**.

Java development tools включают в себя около 40 различных тулов: javac *(компилятор)*, java *(лаунчер для приложений)*, javap *(java class file disassembler)*, jdb *(java debugger)* и др.

2.2 Среда выполнения JRE

Среда выполнения JRE — это пакет всего необходимого для запуска скомпилированной Java-программы. Включает в себя виртуальную машину JVM и библиотеку классов Java — Java Class Library.

JRE **не содержит инструменты для разработки** (компилятор Java, отладчик и т.д). Если вы хотите запустить любую Java программу, вы должны установить JRE.

2.3 JVM

JVM — является сердцем языка программирования Java. Когда мы запускаем программу, JVM несет ответственность за преобразование байт-кода в машинный код. JVM также зависит от платформы и предоставляет основные функции, такие как управления памятью Java, сборкой мусора, и т.д. Мы также можем выделять определенный объем памяти для JVM. JVM является виртуальной машиной, потому что обеспечивает интерфейс, который не зависит от операционной системы и аппаратных средств. Эта независимость от аппаратного обеспечения и операционной системы дает Java-программам возможность выполняться на любом устройстве без необходимости внесения изменений — Write once, run anywhere (Напиши раз — запускай где угодно).

Just-in-time Compiler (JIT) является частью JVM. Он оптимизирует байт-код, уменьшая общее время, необходимое для компиляции байт-кода в машинный код.

2.4 Выполнение кода на JVM

Для того, **чтобы получить код, работающий в JVM**, необходимо выполнить 3 этапа:

- **Загрузка байт-кода и создание экземпляра класса Class**

  Грубо говоря, чтобы попасть на JVM, класс должен быть загружен. Для этого существуют отдельные класс-загрузчики, к ним мы вернемся чуть позже.

- **Связывание или линковка**

  После загрузки класса начинается процесс линковки, на котором байт-код разбирается и проверяется. Процесс линковки в свою очередь происходит в 3 шага:

  1. verification или проверка байт-кода: проверяется корректность инструкций, возможность переполнения стека на данном участке кода, совместимость типов переменных; проверка происходит один раз для каждого класса;
  2. preparation или подготовка: на данном этапе в соответствии со спецификацией выделяется память под статические поля и происходит их инициализация;
  3. resolution или разрешение: разрешение символьных ссылок (когда в байт-коде мы открываем файлы с расширением .class, мы видим числовые значения вместо символьных ссылок).

- **Инициализация полученного объекта Class**

  На последнем этапе класс, который мы создали, инициализируется, и JVM может начинать его исполнение

2.5 Исполнение байт-кода на JVM

В первую очередь, **для исполнения байт-кода**, JVM может его интерпретировать.

Интерпретация — довольно медленный процесс. В процессе интерпретации, интерпретатор "бежит" построчно по класс-файлу и **переводит его в команды, которые понятны JVM**.

Также JVM может его транслировать, т.е. скомпилировать в машинный код, **который будет исполняться непосредственно на CPU**.

Команды, которые исполняются часто, не будут интерпретироваться, а сразу будут транслироваться.

2.6 Компиляция

Компилятор — это программа, **которая преобразует исходные части программ**, написанные на языке программирования высокого уровня, в программу на машинном языке, "понятную" компьютеру.

3 Примитивные типы данных в Java.

Язык Java является языком со **строгой статический типизацией**.

Статическая типизация обозначает, что все переменные и выражения имеют тип, который должен быть известен на этапе компиляции.

Строгая типизация подразумевает строгий контроль соответствия типа переменной типу значения, которое ей может быть присвоено. Также тип переменных и выражений определяет набор и семантику операций, которые можно над ними производить.

Типы данных делятся на **примитивные** и **ссылочные**.

По адресу, связанному с переменной примитивного типа, хранится ее значение. По адресу, связанному с переменной ссылочного типа, хранится ссылка на ее значение. В языке Java существует 8 примитивных типов данных.

Тип	Бит	Диапазон значений	default
byte	8	[-128 ... 127]	0
short	16	[-32768 ... 32767]	0
int	32	[-2147483648 ... 2147483647]	0
long	64	[-9223372036854775808 ... 9223372036854775807]	0L
char	16	['\u0000' ... '\uffff'] или [0 ... 65535]	'\u0000'
float	32	$\pm$[0; $1.4 \cdot 10^{-45}$ ... $3.4028235 \cdot 10^{38}$; ∞], NaN	0.0F
double	64	$\pm$[0; $4.9 \cdot 10^{-324}$ ... $1.7976931348623157 \cdot 10^{308}$; ∞], NaN	0.0
boolean		false; true	false

Для представления значений переменных используются литералы:

- Целые: (префикс) значение (суффикс)
- С плавающей запятой: (префикс) целая_часть . дробная часть основание порядок (суффикс)

Элемент	Символ		Значение
префикс	0		восьмеричный литерал (целый)
	0x	0X	шестнадцатеричный литерал
	0b	0B	двоичный литерал (целый)
суффикс	l	L	литерал типа long
	f	F	литерал типа float
	d	D	литерал типа double
основание	e	E	десятичное основание степени
	p	P	двоичное основание степени

В языке Java диапазоны допустимых значений целочисленных типов **не зависят от машины**, на которой выполняется программа. Это существенно упрощает перенос программного обеспечения с одной платформы на другую.

Числовые значения типа float указываются с суффиксом F, например 3 .14F. А числовые значения с плавающей точкой, указываемые без суффикса F (например, 3 .14), всегда рассматриваются как относящиеся к типу douЬle. Для их представления можно (но не обязательно) использовать суффикс D, например 3 .14D.

Все операции над числами с плавающей точкой производятся по стандарту IEEE 754. В частности, в Java имеются три специальных значения с плавающей точкой.

- Положительная бесконечность
- Отрицательная бесконечность
- Не число (NaN)

Использовать переменную, которой не присвоено никакого значения, нельзя.

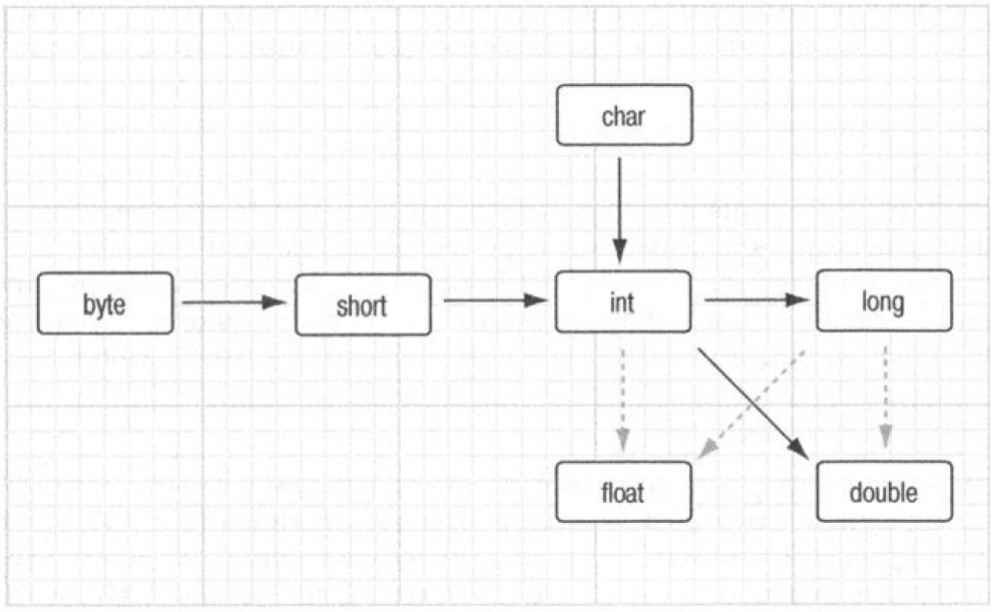


Рис. 1: Преобразование типов данных

Если два числовых значения объединяются бинарной операцией, то перед выполнением операции оба операнда преобразуются в числовые значения одинакового типа по следующим правилам:

- Если хотя бы один из операндов относится к типу double, то и второй операнд преобразуется в тип double.
- Иначе, если хотя бы один из операндов относится к типу float, то и второй операнд преобразуется в тип float.
- Иначе, если хотя бы один из операндов относится к типу long, то и второй операнд преобразуется в тип long.
- Иначе оба операнда преобразуются в тип int.

double → float → long → int

Преобразование числовых типов данных, сопровождающихся потерей данных, называются привидением типов. *Синтаксически задаётся парой скобок, в которой указывается желаемый тип.*

4 Работа с переменными. Декларация. Инициализация. Присваивание.

Существует несколько типов переменных:

- **Поля** — переменные, объявленные в классе;
- **Локальные переменные** — переменные в методе или в блоке кода;
- **Параметры** — переменные в объявлении метода (в сигнатуре).

4.1 Объявление переменной (Declaration)

При объявлении переменной сначала указывается ее тип, а затем имя. Ниже приведен ряд примеров объявления переменных.

Обратите внимание на точку с запятой в конце каждого объявления. Она необходима, поскольку **объявление в языке Java считается полным оператором**, а все операторы в Java завершаются точкой с запятой.

Как упоминалось ранее, в Java **различаются прописные и строчные буквы**. Так, переменные hireday и hireDay считаются разными.

Начиная с версии 1О, объявлять типы локальных переменных необязательно, если их можно вывести из первоначального значения.

4.2 Инициализация переменных

После объявления переменной ее нужно инициализировать с помощью оператора присваивания, поскольку использовать переменную, которой не присвоено никакого значения, **нельзя**.

Чтобы присвоить ранее объявленной переменной какое-нибудь значение, следует указать слева ее имя, поставить знак равенства (=), а справа написать любое допустимое на языке Java выражение, задающее требуемое значение.

При желании переменную **можно объявить и инициализировать одновременно**.

объявление переменной можно размещать в любом месте кода Java

4.3 Работа с переменными

Имя переменной должно состоять из символов Unicode. Оно не должно совпадать с любым из ключевых слов языка Java, а также с логическими константами true и false. Две переменных не могут иметь одинаковые имена, если они находятся в одной области видимости.

Все переменные имеют область видимости имени. Обычно область видимости совпадает с блоком, где объявлена переменная

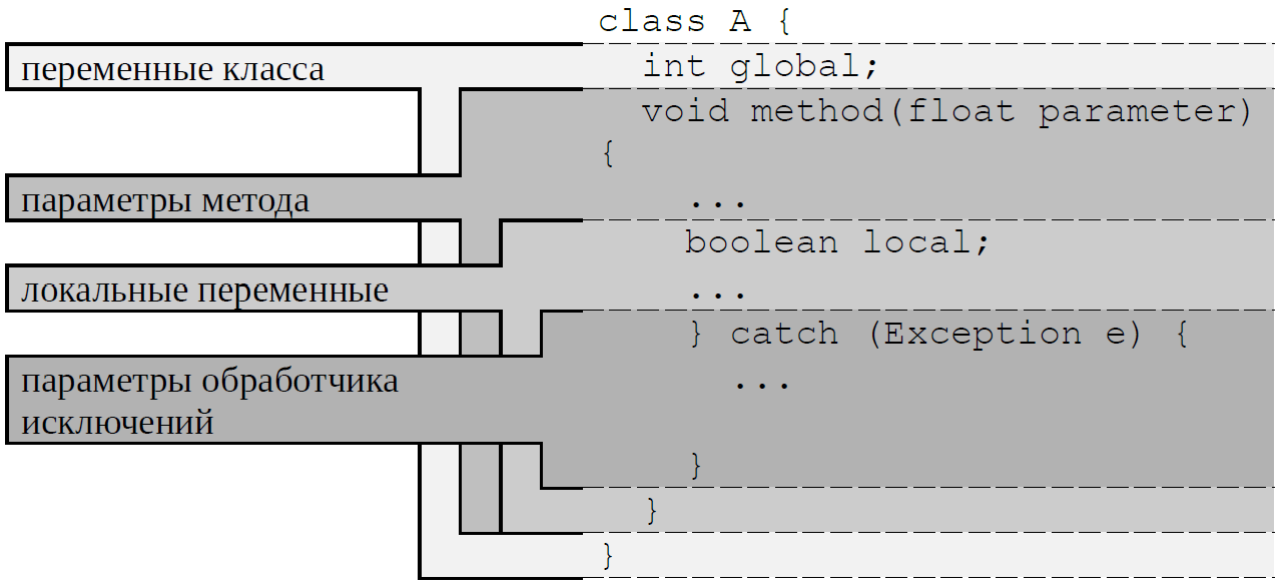


Рис. 2: Преобразование типов данных

5 Инструкции ветвления и циклов

Блок состоит из ряда операторов Java, заключенных в фигурные скобки. Блоки **определяют область видимости** переменных и **могут быть вложенными** один в другой.

В языке Java нельзя объявлять переменные с одинаковым именем в двух вложенных блоках.

5.1 Условные операторы

- Условный оператор if в Java имеет приведенную ниже форму. Условие должно быть заключено в скобки.

if (условие) оператор

Часть **else** данного оператора **не является обязательной и объединяется с ближайшим условным оператором if**:

if (x <= 0) { if (x == 0) sign = O; else sign = -1 ; }

5.2 Неопределенные циклы

Цикл while выполняет выражение (или группу операторов, составляющих блок) до тех пор, пока условие истинно, т.е. оно имеет логическое значение true. Ниже приведена общая форма объявления этого цикла.

while (условие) оператор

Условие цикла while проверяется в самом начале. Следовательно, возможна ситуация, когда код, содержащийся в блоке, образующем тело цикла, вообще не будет выполнен.

Если же требуется, чтобы блок выполнялся хотя бы один раз, проверку условия следует перенести в конец. Эго можно сделать с помощью цикла **do-while**, общая форма объявления которого приведена ниже.

do оператор while (условие)

5.3 Определенные циклы

Цикл **for** является весьма распространенной языковой конструкцией. В нем количество повторений находится под управлением переменной, выполняющей роль счетчика и обновляемой на каждом шаге цикла.

for (инициализания счетчика; условие выполнения тела цикла; порядок обновления счетчика)

5.4 Оператор switch для многовариантного выбора

Языковая конструкция i f/ e 1 s e может оказаться неудобной, если требуется организовать в коде выбор одного из многих вариантов. Для этой цели в Java имеется оператор switch.

```
switch (choice) {
    case 1:
      ...
    break;
    case 2:
      ...
    break;
    default:
      ...
    break;
    }
```

Если не ввести оператор Ьreak в конце ветви case, то возможно последовательное выполнение кода по нескольким ветвям case Очевидно, что такая ситуация чревата ошибками, поэтому пользоваться оператором switch не рекомендуется. Если же вы предпочитаете пользоваться оператором switch в своих программах, скомпилируйте их исходный код с параметром -Xlint: fall through, как показано ниже.

```
javac -Xlint:fallthrough Test.java
```

В этом случае компилятор выдаст предупреждающее сообщение, если альтернативный выбор не завершается оператором break. А если требуется последовательное выполнение кода по нескольким ветвям case, объемлющий метод следует пометить аннотацией @SuppressWarnings ("fall through"). В таком случае никаких предупреждений для данного метода не выдается. Аннотация служит механизмом для предоставления дополнительных сведений компилятору или инструментальному средству, обрабатывающему файлы с исходным кодом Java или классами.

6 Операторы и выражения в Java. Особенности вычисления, приоритеты операций.

В языке Java имеется также тернарная операция ? : , которая иногда оказывается полезной. Ниже приведена ее общая форма:

условие ? выражение_1 : выражение_2

Если условие истинно, то вычисляется первое выражение, а если оно ложно второе выражение.

постфиксные операторы	*expr*++ *expr*-- . ::
унарные операторы	++*expr* --*expr* +*expr* -*expr* ~ !
операции с типами	new (*cast*)*expr*
умножение/деление	* / %
сложение/вычитание	+ -
операторы сдвига	<< >> >>>
операторы отношения	< > <= >= instanceof
операторы равенства	== !=
поразрядное И	&
поразрядное искл. ИЛИ	^
поразрядное ИЛИ	|
логическое И	&&
логическое ИЛИ	||
условный оператор	? :
операторы присваивания	= += -= *= /= %= >>= <<= >>>= &= ^= |=
стрелка лямбда	->

Рис. 3: Преобразование типов данных

7 Математические функции в составе стандартной библиотеки Java. Класс java.lang.Math.

В состав класса Math входит целый набор математических функций, которые нередко требуются для решения практических задач. В частности, чтобы извлечь квадратный корень из числа, применяется метод sqrt():

В языке Java не поддерживается операция возведения в степень. Для этой цели следует вызвать **метод pow()** из класса Math.

Рассмотрим в качестве примера выражение n % 2. Как известно, если число n является четным, то результат данной операции равен О, а иначе — l. Если же число n оказывается отрицательным, то результат данной операции будет равен -1.

Метод floorMod() предназначен для решения давней проблемы с остатками от целочисленного деления.

((положение+ коррекция) % 12 + 12) % 12
floorMod(положение + коррекция, 12)

В состав класса Math входят также перечисленные ниже методы для вычисления обычных тригонометрических функций.

Кроме того, в него включены методы для вычисления экспоненциальной и обратной к ней логарифмической функции (натурального и десятичного логарифмов):

И, наконец, в данном классе определены также следующие две константы как приближенное представление чисел п и е.

7.1 Класс java.lang.Math.

При вызове методов для математических вычислений класс math можно не указывать явно, включив вместо этого в начало исходного файла следующую строку кода:

import static java.lang.Math.*;

В классе **Math** предоставляется ряд методов, обеспечивающих более надежное выполнение арифметических операций. Математические операции негласно возвращают неверные результаты, когда вычисление приводит к переполнению. Например, результат умножения одного миллиарда на три **[11000000000 * З]** оказывается равным -1294967296, поскольку максимальное значение типа int лишь ненамного превышает два миллиарда. А если вместо этого вызвать метод **Math.multiplyExact (1000000000, 3)**, то будет сгенерировано исключение. Это исключение можно перехватить, чтобы нормально завершить выполнение программы, а не позволить ей продолжаться негласно с неверным результатом умножения. В классе **Math** имеются также методы **addExact (), suЬtractExact (), incrementExact (), decrementExact (), negateExact ()** с параметрами типа **int** и **long**.

8 Форматированный вывод числовых данных.

Числовое значение x можно вывести на консоль с помощью выражения System. out. println (x). В результате на экране отобразится число с **максимальным количеством значащих цифр, допустимых для данного типа**.

Метод printf () позволяет задавать произвольное количество параметров.

Каждый спецификатор формата, начинающийся с символа %, заменяется **соответствующим параметром**. А символ преобразования, которым завершается спецификатор формата, **задает тип форматируемого значения**.

Символ преобразования	Тип	Пример
`d`	Десятичное целое	`159`
`x`	Шестнадцатеричное целое	`9f`
`f`	Число с фиксированной или плавающей точкой	`15.9`
`e`	Число с плавающей точкой в экспоненциальной форме	`1.59e+01`
`g`	Число с плавающей точкой в общем формате (чаще всего используется формат `e` или `f`, в зависимости от того, какой из них дает более короткую запись)	—
`a`	Шестнадцатеричное представление числа с плавающей точкой	`0x1.fccdp3`
`s`	Символьная строка	`Hello`
`c`	Символ	`H`
`b`	Логическое значение	`true`
`h`	Хеш-код	`42628b2`
`tx` или `Tx`	Дата и время (`T` означает обозначение даты и времени прописными буквами)	Устарел, пользуйтесь классами из пакета `java.time`, как поясняется в главе 6 второго тома настоящего издания
`%`	Знак процента	`%`
`n`	Разделитель строк, зависящий от платформы	—

Рис. 4: Символы преобразования для метода printf()

Флаг	Назначение	Пример
`+`	Выводит знак не только для отрицательных, но и для положительных чисел	`+3333.33`
Пробел	Добавляет пробел перед положительными числами	`| 3333.33|`
`0`	Выводит начальные нули	`003333.33`
`-`	Выравнивает поле по левому краю	`|3333.33 |`
`(`	Заключает отрицательные числа в скобки	`(3333.33)`
`,`	Задает использование разделителя групп	`3,333.33`
`#` (для формата `f`)	Всегда отображает десятичную точку	`3,333.`
`#` (для формата `x` или `o`)	Добавляет префикс `0x` или `0`	`0xcafe`
`$`	Определяет индекс параметра, предназначенного для форматирования. Например, выражение `%1$d %1$x` указывает на то, что первый параметр должен быть сначала выведен в десятичной, а затем в шестнадцатеричной форме	`159 9F`
`<`	Задает форматирование того же самого значения, которое отформатировано предыдущим спецификатором. Например, выражение `%d %<x` указывает на то, что одно и то же значение должно быть представлено как в десятичной, так и в шестнадцатеричной форме	`159 9F`

Рис. 5: Флаги для метода printf()

Для составления отформатированной символьной строки **без последующего ее вывода** можно вызвать статический метод **String.format()**,как показано ниже.