АВТОР

Aditi Nawghare

Инженер-программист в **Siemens**

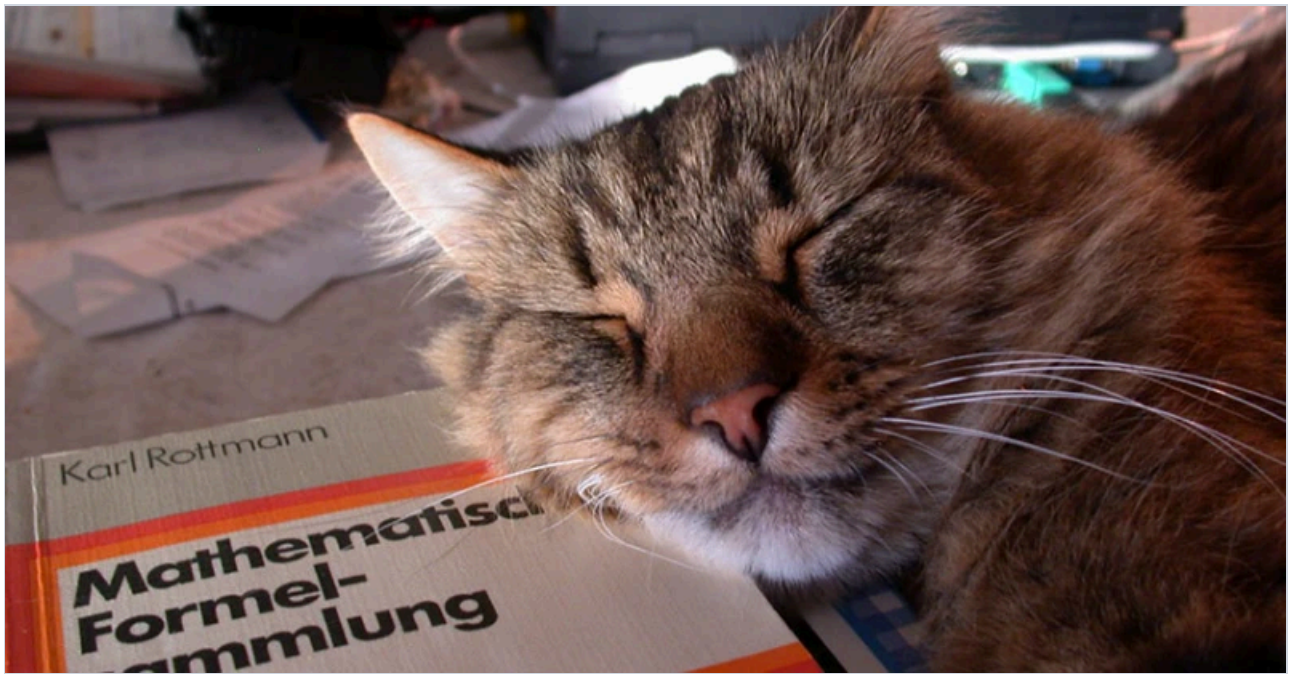
21 октября 2020 279492 43

Форматируем вывод чисел в Java

Статья из группы Архив info.javarush

Всем привет!

Часто в наши методы приходят числа, которые нужно отобразить в каком-то особом формате. Вроде бы как мелочь, но как бы вы реализовали эту задачу?



Предлагаем над этим сегодня немного поразмыслить.

Для начала, чтобы с головой окунуться в форматирование чисел в Java, давайте вспомним метод `format` класса

`String`:

`public static String format(String format, Object… args)` — возвращает строку, отформатированную из строки `format` с помощью остальных аргументов `args`.

И сразу пример:

```
1 String str = String.format("Привет - %s! Как дела %s?", "Саша", "на работе");
2 System.out.println(str);
```

В итоге мы получим вывод в консоли:

```
Привет - Саша! Как дела на работе?
```

Методы printf и format

`String.format()` — не единственный метод для форматирования строки. Его аналогами могут служить `System.out.printf()` и `System.out.format();`.

Так, предыдущий код мы можем заменить на:

```
1 System.out.printf("Привет - %s! Как дела %s?", "Саша", "на работе");
```

или

```
1 System.out.format("Привет - %s! Как дела %s?", "Саша", "на работе");
```

Вывод в консоли при этом останется тем же. Единственным отличием служит только то, что данные методы сразу выводят значение в консоли, в отличие от `String.format()`.

Но мне `String.format()` нравится больше, так как нам не всегда нужно выводить результат в консоли, поэтому далее мы будем использовать именно этот способ.

Вернемся же к нашему примеру. Что мы видим? А то, что в места, где были символы — `%s`, вставлены строки — `"Саша"` и `"на работе"`.

Каким образом это может нам помочь при разработке?

Представьте, что у вас есть большой шаблонный текст, но в некоторых местах вам нужно вставлять значения, которые могут быть разными и приходить в качестве аргументов извне. Вот тут нам и пригодится данное форматирование.

Спецификаторы формата начинаются со знака процента % и заканчиваются символом, указывающим тип аргумента, который нужно отформатировать.

И, как вы наверное поняли, `%s` используется для вставки объектов — строк.

Но если мы попробуем вставить, к примеру, `double` в место, в котором прописан объект строки:

```
String str = String.format("Привет - %s! Как дела %s?", 55.6, "на работе");
```

это также сработает. `double` будет приведен к строке, и мы получим:

```
Привет - 55.6! Как дела на работе?
```

Помимо строк и чисел с плавающей запятой, в Java есть и другие типы, не так ли? Поэтому давайте взглянем на весь арсенал:

	Тип форматируемого значения	Пример

%s	Любой тип, который будет приведен к строке	`String.format("Привет %s!","мир")`
		Результат:
		`Привет мир!`

%b	Любой тип, который будет приведен к `boolean`: true — если значение не null, false — если null	`String.format("Привет %b!",null)`
		Результат:
		`Привет false`

%h	Можно передавать любой объект, который будет приведен к шестнадцатеричной строке значения из метода `hashCode ()`	`String.format("Привет %h!","мир")`
		Результат:
		`Привет 106c44!`

%c	Используется для задания символа Unicode (**char**)	`String.format("Привет м%ср!",'и')`
		Результат:
		`Привет мир!`

%d	Задается целое число (**int. byte, short, int, long, BigInteger**)	`String.format("Мне уже %d!",20)`
		Результат:
		`Мне уже 20!`

%f	Используется для задания числа с плавающей запятой	`String.format("Число ПИ равно - %f!", 3.14159)`
		Результат:
		`Число ПИ равно - 3,141590!`

%e	Числа с плавающей запятой в экспоненциальном представлении	`String.format("Число ПИ равно - %e!", 3.14159);`
		Результат:
		`Число ПИ равно - 3,141590e+00!`

%a	Числа с плавающей запятой будут представлены в	`String.format("Число ПИ равно - %a!", 3.14159)` Результат:

	шестнадцатеричном виде	`Число ПИ равно - 0x1.921f9f01b866ep1!`

%x	Передается целое число (**int. byte, short, int, long, BigInteger**), результатом форматирования будет символ под данным номером в таблице ASCII	`String.format("Мне уже %x!",25)`
		Результат:
		`Мне уже 19!`

%o	Принимается целое число (**int. byte, short, int, long, BigInteger**), которое будет представлено в виде восьмеричного числа	`String.format("Мне уже %o!",25);`
		Результат:
		`Мне уже 31!`

%t	Префикс для преобразований даты и времени. Для форматирования требуются дополнительные флаги	`String.format("Сегодня %tA",new Date())`
		Результат:
		`Сегодня суббота`

%n	Разделитель строк для конкретной платформы. Аналог \n	`String.format(" Привет %n Привет")`
		Результат:
		`Привет Привет`

Давайте используем для `double` более подходящий формат:

```
String str = String.format("Расстояние от Киева до Одессы - %f. Не так уж и мало, не правда
System.out.println(str);
```

Вывод в консоль:

```
Расстояние от Киева до Одессы - 475,400000. Не так уж и мало, не правда ли?
```

Как вы уже поняли, `%f` будет более подходящим спецификатором для чисел с плавающей запятой, которые

включают в себя такие типы данных как `double` и `float` в Java.

С этим спецификатором мы можем форматировать число с плавающей запятой:

```
String str = String.format("Расстояние от Киева до Одессы - %.2f. Не так уж и мало, не прав
```

Вставка `.2` в данный спецификатор обрежет количество знаков после запятой до двух, и мы получим вывод:

```
Расстояние от Киева до Одессы - 475,40. Не так уж и мало, не правда ли?
```

`.2` — не единственная поднастройка спецификаторов. Комбинация данных поднастроек называется **инструкцией**.

Общий вид **инструкции** такой:

```
%[аргумент_индекс][флаги][ширина][.точность]спецификатор типа
```

А теперь расшифруем все по порядку:

- **[аргумент_индекс]** — целое число, указывающее позицию в списке аргументов. К примеру, ссылка на первый аргумент 1$, ссылка на второй аргумент — 2$, и т.д. Если же позиция не была задана, аргументы должны находиться в том же порядке, что и ссылки на них в строке форматирования.
- **[флаги]** — специальные символы для форматирования. Например:
  - `+` флаг, означающий, что если числовое значение положительное, оно должно включать знак +
  - `-` означает выравнивание результата по левому краю
  - `,` устанавливает разделитель тысяч у целых чисел
- **[ширина]** — положительное целое десятичное число, определяющее минимальное количество символов, которые будут выведены. Если перед этим числом стоит 0, то недостающие символы будут дополнены 0, если 0 нет, то пробелами.
- **[.точность]** — неотрицательное целое число с точкой перед ним. Как правило используется для ограничения количества символов. Специфика поведения зависит от конкретного вида спецификатора.

Также хотелось бы отметить, что все вышеперечисленные элементы инструкции не обязательны, и всё будет работать и без них.

В качестве примера использования поднастроек представим, что нам нужен специфический вывод числа Пи:

```
String str = String.format("%1$+09.5f", 3.1415926535897);
System.out.print(str);
```

И соответственно, вывод в консоли:

```
+03,14159
```

Вроде несложно, так?

Но когда заходит речь о форматировании числа, то нельзя обойти стороной **DecimalFormat**. Давайте разберемся, что имеется в виду.

DecimalFormat

DecimalFormat — класс для форматирования любого числа в Java, будь то целое число или число с плавающей запятой.

Когда происходит создание объекта `DecimalFormat`, прямо в конструкторе можно задать шаблон форматирования приходящих чисел.

Как будет выглядеть наш пример с использованием `DecimalFormat`:

```
1 DecimalFormat dF = new DecimalFormat( "#.###" );
2 double value = 72.224463;
3 System.out.print(dF.format(value));
```

Вывод в консоли:

```
72,224
```

Строка `#.###` является шаблоном, который указывает, что мы форматируем передаваемое значение до 3 десятичных знаков.

Какие ещё доступны символы для шаблонов? Вот некоторые из них:

- `#` — цифра, ведущие нули опускаются;
- `0` — цифра отображается всегда, даже если в номере меньше цифр (в таком случае отображается 0);
- `.` — знак десятичного разделителя;
- `,` — знак группировки разделителей (например, разделитель тысяч);
- `;` — разделяет форматы;
- `-` — отмечает префикс отрицательного числа;
- `%` — умножает на 100 и показывает число в процентах;
- `?` — умножает на 1000 и показывает число в промилле;
- `E` — разделяет мантиссу и порядок для экспоненциального формата.

Давайте взглянем на несколько примеров:

```
1 System.out.println(new DecimalFormat( "###,###.##" ).format(74554542.224463));
```

Вывод в консоли:

```
74 554 542,22
```

```
1 System.out.println(new DecimalFormat( "%###.##" ).format(0.723456));
```

Вывод в консоли:

```
%72,35
```

```
System.out.println(new DecimalFormat( "000.###" ).format(42.224463));
```

Вывод в консоли:

```
042,224
```

Не обязательно создавать каждый раз новый объект `DecimalFormat`, чтобы задать новый шаблон. Будет достаточно использовать его методы `applyPattern` и `applyLocalizedPattern`:

```
DecimalFormat dF = new DecimalFormat("###.###");
dF.applyPattern("000000.000");
dF.applyLocalizedPattern("#,#00.0#");
```

Когда мы говорим о форматировании числа с плавающей запятой, нас немало интересует округление, не так ли?

Так вот, при обрезании числа со знаками после запятой, выходящими за заданный шаблон, `DecimalFormat` округляет число в большую сторону, если последнее обрезаемое число больше 5.

А если последнее обрезаемое — 5? Ведь в таком случае это число ровно посередине между ближайшими целыми.

В этом случае в расчет берется предыдущее до него число.

Если предыдущее число чётное, округление производится:

```
DecimalFormat dF = new DecimalFormat("##.###");
String result = dF.format(56.4595);
System.out.println((result));
```

Вывод в консоли:

```
56,459
```

Если нечётное — не производится:

```
DecimalFormat dF = new DecimalFormat("##.###");
String str = dF.format(56.4595);
System.out.println((str));
```

Вывод в консоли:

```
56,459
```

Разницей между форматированием чисел с плавающей запятой с использованием `String.format()` и `DecimalFormat.format()` можно считать то, что в первом случае будут присутствовать конечные нули, даже если нет дробной части.

Например:

```
String firstStr = String.format("%.4f", 9.00001273);
System.out.println((firstStr));
```

Вывод в консоли:

```
9,0000
```

```
DecimalFormat decimalFormat = new DecimalFormat("#.####");
String secondStr = decimalFormat.format(9.00001273);
System.out.println((secondStr));
```

Вывод в консоли:

```
9
```

Как видите, при форматировании числа **9.00001273** с точностью до четырех десятичных разрядов метод `format()` у класса `String` выведет значение **9.0000**, при этом у `DecimalFormat` аналогичный метод `format()` выведет **9**.

BigDecimal и BigInteger

Раз мы затронули такую тему округления чисел в Java, поговорим и о том, как для таких операций использовать класс `BigDecimal`.

Этот класс ориентирован на работу с действительно БОЛЬШИМИ числами: для него максимальные значения `double` и `float` слишком малы. У этого класса есть много различных настроек для округления числа с плавающей запятой, а также много методов для арифметических операций.

У него есть похожий класс, но ориентированный на работу с БОЛЬШИМИ целыми числами — `BigInteger`.

Подробнее о `BigDecimal` и `BigInteger` можно почитать в этой статье.

Форматирование Date и Time

Выше только упоминалось, что с помощью `format()` класса `String` можно еще и форматировать время и дату.

Что же, давайте взглянем, как это делается.

Во-первых, хотелось бы напомнить, что для дат используется спецификатор формата `%t`.

Во-вторых, при форматировании шаблона, для каждого спецификатора формата для дат требуются дополнительные флаги форматирования.

Вот возможные флаги форматирования для дат:

Флаги	Описание
%tB	Полное название месяца, например, January, February и т.д.
%tb	Сокращенное название месяца, например, Jan, Feb и т.д.
%tA	Полное название дня недели, например, Sunday, Monday
%ta	Сокращенное название дня недели, например, Sun, Mon и т.д.
%tY	Год в формате 4 цифры, например, от 0000 до 9999
%ty	Год в формате 2 цифры, например, от 00 до 99
%tm	Месяц отформатирован с нуля в начале, например, от 01 до 12
%tc	Дата и время в формате %ta %tb %td %tT %tZ %tY, например, Mon Feb 17 03:56:12 PST 2020
%tD	Дата в формате %tm/%td/%ty
%td	День месяца в формате двух цифр, например, от 01 до 31
%te	День месяца в формате без 0 в начале, например от 1 до 31
%tT	Время в 24-часовом формате, например, %tH:%tM:%tS
%tH	Час дня в 24-часовом формате, от 00 до 23
%tI	Час дня для 12-часового формата, например, от 01 до 12
%tM	Минуты в часе форматируются с нуля в начале, например, от 00 до 59
%tS	Секунды в минуте, состоящие из двух цифр, например, от 00 до 59
%tZ	Аббревиатура часового пояса, например, PST, UTC и т.д.

Это сокращенный список возможных флагов форматирования дат — их очень много, на любой вкус. Полный список их как и возможных спецификаторов можно посмотреть по этой ссылке.

Давайте рассмотрим, как этим пользоваться. В этот раз используем не `String.format()`, а сразу `System.out.printf()`.

Пример 1

Помимо всего, зададим язык результа, передав его первым аргументом в метод:

```
1 Date date = new Date();
2 System.out.printf(Locale.ENGLISH,"%tB %te, %tY",date,date,date);
```

Вывод в консоли:

```
October 11, 2020
```

Без задавания языка будет использован язык по умолчанию (к примеру, у меня он русский).

Пример 2

Давайте выведем на экран более полную дату:

```
1 Date date = new Date();
2 System.out.printf("%td %tB %tY года %n%tH:%tM:%tS",date,date,date,date,date,date,date);
```

И вывод в консоли:

```
11 октября 2020 года
13:43:22
```

Столько раз передавать аргументом один и тот же объект Date... Как-то выглядит не очень, не так ли?

Давайте воспользуемся внутренней поднастройкой `$` для указания аргумента, который мы хотим использовать:

```
1 System.out.printf("%1$td %1$tB %1$tY года %n%1$tH:%1$tM:%1$tS",date);
```

Вывод в консоли у нас и не изменится.

Есть и другие не менее интересные способы форматирования даты. О них и немного подробнее о времени и дате в Java можно почитать вот в этом материале.

На этом у меня на сегодня всё, спасибо за внимание!