

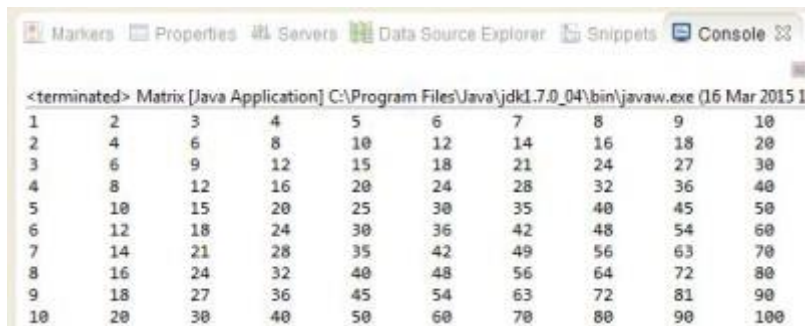
Источник: <http://study-java.ru/uroki-java/formatirovanie-chisel-i-texta-v-java/#more-864>

Для начала немного мотивации. Рассмотрим пример, в котором рассчитывается и выводится на консоль таблица умножения:

```
int[][] multiplyTab = new int[10][10];

for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        multiplyTab[i][j] = (i+1)*(j+1);
        //вывод ряда чисел разделенных знаком табуляции
        System.out.print(multiplyTab[i][j] + "\t");
    }
    System.out.println();
}
```

В данном случае для разделения чисел мы использовали знак табуляции, это дало следующий результат:



<terminated> Matrix [Java Application] C:\Program Files\Java\jdk1.7.0_04\bin\javaw.exe (16 Mar 2015 11:00:00 AM)

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

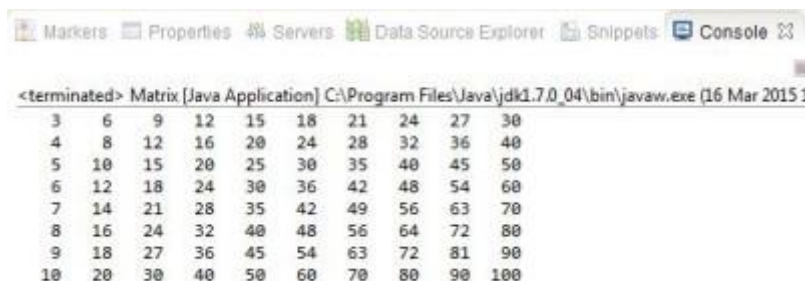
Таблица выглядит ровно, но она слишком широкая. Для того, чтобы сделать таблицу более компактной, будем использовать метод **printf()**. Заменяем в предыдущем коде строку

```
System.out.print(multiplyTab[i][j] + "\t");
```

на строку

```
System.out.printf("%4d", multiplyTab[i][j]);
```

Получим следующий результат:



<terminated> Matrix [Java Application] C:\Program Files\Java\jdk1.7.0_04\bin\javaw.exe (16 Mar 2015 11:00:00 AM)

3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Как мы видим, таблица стала компактнее. Более того, теперь мы можем уменьшать или увеличивать промежутки между числами по нашему желанию. Для этого нужно лишь заменить число **4** в выражении «**%4d**».

Далее рассмотрим метод ***printf()*** и его возможности подробнее.

Использование ***printf*** для форматирования в Java

Метод ***printf()*** принадлежит классу ***PrintStream***, который отвечает за вывод информации. Уже знакомые нам методы ***print()*** и ***println()*** также являются методами класса ***PrintStream***.

Метод ***printf*** определен следующим образом:

```
printf(String format, Object... args)
```

Первый аргумент ***format*** это строка, определяющая шаблон, согласно которому будет происходить форматирование. Для ее записи существуют определенные правила.

В предыдущем примере формат был «***%4d***», где ***d*** означает вывод десятичного целого числа, а ***4*** — означает то, что если количество знаков в числе меньше, чем 4, то оно будет спереди дополнено пробелами на недостающее (до 4-х) количество знаков (тем самым подвинуто вправо).

Для наглядности приведем еще один пример, который выводит столбиком несколько чисел.

```
System.out.printf("%6d%n%6d%n%6d%n%6d%n%6d%n%6d", 666666, 55555, 4444, 333, 22, 1);
```

На консоль будет выведено:

```
666666
 55555
  4444
   333
    22
     1
```

Здесь в строке форматирования мы использовали несколько раз ***%6d%n***, где каждое ***%6d*** задает формат для одного из чисел, указанных далее в качестве аргументов. Первое ***%6d*** форматирует число **666666**, второе ***%6d*** форматирует **55555** и т.д. ***%n*** означает перевод строки. Поскольку ко всем числам было применено форматирование ***%6d***, то числа, которые содержат менее 6 знаков подвинуты вправо на недостающее количество знаков и тем самым красиво выровнены.

Данный пример иллюстрирует также то, что метод ***printf*** может содержать несколько аргументов после строки с форматом. На что указывает ***Object... args*** в сигнатуре метода. Эти аргументы должны соответствовать ссылкам на них в строке формата. Например, если в строке формата стоит символ ***d***, отвечающий за вывод целого десятичного числа, а далее в аргументе вы укажете строку, при компиляции произойдет ошибка преобразования формата `java.util.IllegalFormatConversionException`. Если аргументов больше, чем определено в формате, то лишние аргументы будут игнорироваться.

Общий вид инструкции форматирования следующий:

`%[argument_index$][flags][width][.precision]conversion`

- **%** — специальный символ, обозначающий начало инструкций форматирования.
- **[argument_index\$]** — целое десятичное число, указывающее позицию аргумента в списке аргументов. Ссылка на первый аргумент "1\$", ссылка на второй аргумент "2\$" и т.д. Не является обязательной частью инструкции. Если позиция не задана, то аргументы должны находиться в том же порядке, что и ссылки на них в строке форматирования.
- **[flags]** — специальные символы для форматирования. Например, флаг "+" означает, что числовое значение должно включать знак +, флаг "-" означает выравнивание результата по левому краю, флаг «,» устанавливает разделитель тысяч у целых чисел. Не является обязательной частью инструкции.
- **[width]** — положительное целое десятичное число, которое определяет минимальное количество символов, которые будут выведены. Не является обязательной частью инструкции.
- **[.precision]** — не отрицательное целое десятичное число с точкой перед ним. Обычно используется для ограничения количества символов. Специфика поведения зависит от вида преобразования. Не является обязательной частью инструкции.
- **conversion** — это символ, указывающий, как аргумент должен быть отформатирован. Например **d** для целых чисел, **s** для строк, **f** для чисел с плавающей точкой. Является обязательной частью инструкции.

Все возможные флаги и преобразования (**conversion**) указаны в [официальной документации](#). Здесь мы не ставим цель изучить их все, наша цель — научиться применять форматирование. Поэтому рассмотрим несколько примеров.

Пример 1. Наглядный пример инструкции форматирования в ее полном виде приведен на следующей картинке:

%	1\$	+0	20	.10	f
Begin Format Specifier	Argument Index	Flags	Width	Precision	Conversion

Если мы применим формат с картинки к числу `Pi`

```
System.out.printf("%1$+020.10f", Math.PI);
```

Получим следующий результат на консоли:

```
+000000003,1415926536
```

Разберем данную инструкцию с конца:

- **f** — указывает на то, что выводим число с плавающей точкой.
- **.10** — выведенное число будет содержать 10 знаков после запятой.
- **20** — всего выведенное число будет иметь ширину в 20 символов

- **+0** — недостающие (до 20-ти) символы будут заполнены нулями, перед числом будет указан знак (+)
- **1\$** — данный формат применяется к первому аргументу, который находится после строки форматирования. В данном примере это было указывать не обязательно.

Пример 2. Рассмотрим пример демонстрирующий вывод на консоль **до** знакомства с форматированием и **после**.

Без форматирования

```
Integer i=675;  
double root;  
root = Math.sqrt(i);  
System.out.println("Корень числа " + i + " равен " + root );
```

На консоль будет выведено:

Корень числа 675 равен 25.98076211353316

В этом случае преобразование кода в строку автоматически генерируется компилятором Java. Этот способ плох тем, что при большом количестве переменных и текста для вывода, легко потерять контроль над результатами.

С форматированием

```
Integer i=675;  
double root;  
root = Math.sqrt(i);  
System.out.printf("Корень числа %d равен %f", i, root );
```

Где **%d** отвечает за вывод значения переменной **i**, а **%f** за вывод значения переменной **root**. При этом мы уже не используем конкатенацию.

На консоль будет выведено:

Корень числа 675 равен 25,980762

Как мы видим, формат автоматически округляет число до 6 знаков после запятой. Однако, при форматировании мы можем устанавливать такое количество знаков после запятой, которое нам нужно, например:

```
System.out.printf("Корень числа %d равен %.2f", i, root );
```

Выведет число с двумя знаками после запятой.

Далее на примерах рассмотрим наиболее популярные правила форматирования.

Форматирование целых чисел

Вывод целого числа

```
System.out.printf("%d", 7845); // --> "7845"
```

Вывод целого числа с разделением тысяч

```
System.out.printf("%d", 7845); // --> "7 845"
```

Число менее 7 знаков будет «подвинуто» вправо на недостающее количество знаков.

```
System.out.printf("%7d", 7845); // --> "    7845"
```

Число менее 7 знаков будет дополнено нулями слева на недостающее количество знаков.

```
System.out.printf("%07d", 7845); // --> "0007845"
```

Число будет дополнено знаком + и, если оно менее 7 знаков, то будет дополнено нулями на недостающее количество знаков.

```
System.out.printf("%+07d", 7845); // --> "+007845"
```

Число будет выровнено по левому краю и, если оно менее 7 знаков, то будет дополнено пробелами справа на недостающее количество знаков.

```
System.out.printf("%-7d", 7845); // --> "7845   "
```

Разумеется вместо 7 можно использовать любое другое положительное целое число.

Форматирование чисел с плавающей точкой

Вывод числа e. Автоматическое округление до 6 знаков после запятой.

```
System.out.printf("%f", Math.E); // --> "2,718282"
```

Число менее 10 знаков будет «подвинуто» вправо на недостающее количество знаков.

```
System.out.printf("%10f", Math.E); // --> "    2,718282"
```

Число менее 10 знаков будет дополнено нулями слева на недостающее количество знаков.

```
System.out.printf("%010f", Math.E); // --> "0002,718282"
```

Число будет дополнено знаком + и, если оно менее 10 знаков, то будет дополнено нулями на недостающее количество знаков.

```
System.out.printf("%+010f", Math.E); // --> "+002,718282"
```

Число будет выведено с 15 знаками после запятой.

```
System.out.printf("%.15f", Math.E); // --> "2,718281828459045"
```

Число будет выведено с 3-мя знаками после запятой и, если оно менее 8 символов, то будет «подвинуто» вправо на недостающее количество знаков.

```
System.out.printf("%8.3f", Math.E); // --> "    2,718"
```

Число будет выровнено по левому краю, выведено с 3-мя знаками после запятой и, если оно менее 8 знаков, то будет дополнено пробелами справа на недостающее количество знаков.

```
System.out.printf("%-8.3f", Math.E); // --> "2,718    "
```

Форматирование строк

Вывод строки.

```
System.out.printf("%s\n", "Hello"); // --> "Hello"
```

Если строка содержит менее 10 символов, то «подвинуть» ее вправо на недостающее количество символов.

```
System.out.printf("%10s\n", "Hello"); // --> "        Hello"
```

Строка будет выровнена по левому краю и, если она менее 10 символов, то будет дополнена справа пробелами на недостающее количество символов.

```
System.out.printf("%-10s\n", "Hello"); // --> "Hello      "
```

Выведет первые 3 символа строки.

```
System.out.printf("%.3s\n", "Hello"); // --> "Hel"
```

Выведет первые 3 символа строки и подвинет их вправо на недостающее до 8 количество символов.

```
System.out.printf("%8.3s\n", "Hello"); // --> "        Hel"
```

Локализация

В разных странах некоторые записи принято производить по-разному. Например, в одних странах дробное число принято писать с точкой «3.68», а в других с запятой «3,68». Java нам позволяет соблюдать эти традиции. Метод **printf** имеет еще одну сигнатуру:

printf(Locale l, String format, Object... args)

Первым аргументом стоит **Locale l**, который и определяет локализацию. Для того, чтобы использовать локализацию необходимо вначале файла с вашим кодом импортировать библиотеку `java.util.Locale`.

```
import java.util.Locale;
```

Рассмотрим несколько примеров применения:

```
System.out.printf(Locale.ENGLISH,"%d\n", 1000000 );// 1,000,000
System.out.printf(Locale.GERMAN,"%d\n", 1000000 ); // 1.000.000
System.out.printf(Locale.FRANCE,"%d\n", 1000000 ); // 1 000 000
```

В зависимости от указанной страны используются разные разделители для тысяч.

```
System.out.printf(Locale.ENGLISH,"%0.2f\n", 9.87 ); //9.87
System.out.printf(Locale.FRANCE,"%0.2f\n", 9.87 ); //9,87
```

В зависимости от указанной страны используются разные разделители у дробных чисел.

Использование String.format

В случае, если нет необходимости выводить отформатированную строку, а нужно просто ее сохранить для дальнейшего использования (например, для записи в лог или базу данных) следует использовать метод **format** из класса **String**. Принципы форматирования в этом случае абсолютно такие же, как у описанного выше **printf**, но этот метод вместо вывода строки сохраняет ее как отформатированную строку.

Пример использования:

```
String s = String.format("Курс валют: %-4s%-8.4f  %-4s%-8.4f", "USD", 58.4643, "EUR", 63.3695);
```

Это далеко не все возможности форматирования в Java. Существуют несколько классов, предназначенных для более сложного форматирования, но их оставим для дальнейших уроков.

Закончить урок хочется примером, в котором используются форматирование всех изученных в этом уроке типов данных: целых чисел, чисел с плавающей точкой и строк.

Пример: Таблица курсов валют

```
System.out.printf("%-5s%-11s%-25s%-11s%n", "Код", "За единиц", "Валюты", "Рублей РФ");  
System.out.println("-----");  
System.out.printf("%-5s%-11d%-25s%-11.4f%n", "AUD", 1, "Австралийский доллар", 44.9883);  
System.out.printf("%-5s%-11d%-25s%-11.4f%n", "GBP", 1, "Фунт стерлингов", 86.8429);  
System.out.printf("%-5s%-11d%-25s%-11.4f%n", "BYR", 10000, "Белорусский рубль", 39.7716);  
System.out.printf("%-5s%-11d%-25s%-11.4f%n", "DKK", 10, "Датская крона", 84.9192);  
System.out.printf("%-5s%-11d%-25s%-11.4f%n", "USD", 1, "Доллар США", 58.4643);  
System.out.printf("%-5s%-11d%-25s%-11.4f%n", "EUR", 1, "Евро", 63.3695);  
System.out.printf("%-5s%-11d%-25s%-11.4f%n", "KZT", 100, "Казахский тенге", 31.4654);
```

Результат вывода на консоль:

Код	За единиц	Валюты	Рублей РФ
AUD	1	Австралийский доллар	44,9883
GBP	1	Фунт стерлингов	86,8429
BYR	10000	Белорусский рубль	39,7716
DKK	10	Датская крона	84,9192
USD	1	Доллар США	58,4643
EUR	1	Евро	63,3695
KZT	100	Казахский тенге	31,4654

Полезные ссылки из официальной документации:

- <http://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html>
- <https://docs.oracle.com/javase/tutorial/java/data/numberformat.html>
- <http://docs.oracle.com/javase/tutorial/essential/io/formatting.html>

Спецификаторы формата Java

Вот краткий справочник по всем поддерживаемым спецификаторам.

Спецификатор	Тип	Вывод
%a	floating point (except BigDecimal)	Шестнадцатеричный вывод числа с плавающей запятой
%b	Any type	“true” если не 0, “false” если 0
%c	character	Юникод
%d	integer (incl. byte, short, int, long, bigint)	Десятичное целое
%e	floating point	десятичное число в научной записи
%f	floating point	десятичное число
%g	floating point	десятичное число, возможно, в научных обозначениях в зависимости от точности и значения.
%h	any type	Шестнадцатеричная строка значения из метода hashCode ().
%n	none	Специфичный для платформы разделитель строк.
%o	integer (incl. byte, short, int, long, bigint)	Восьмеричное число
%s	any type	Строковое значение
%t	Date/Time (incl. long, Calendar, Date and TemporalAccessor)	% t — это префикс для преобразования даты / времени.
%x	integer (incl. byte, short, int, long, bigint)	Шестнадцатеричная строка.

Форматирование даты и времени Java

Использование символов форматирования с «%T» вместо «%t» приводит к выводу заглавных букв.

- %tA Полное название дня недели, например, «Воскресение понедельник»
- %ta Сокращенное название дня недели, например, «Солнце», «Пн» и др.
- %tB Полное название месяца, например «Январь», «Февраль» и др.
- %tb Сокращенное название месяца, например «Январь», «февраль» и т. Д.
- %tC Century часть года, отформатированная двумя цифрами, например, «00» — «99».
- %tc Дата и время в формате «% ta% tb% td% tT% tZ% tY», например «Пт 17 февраля 07:45:42 PST 2017»
- %tD Дата в формате «% tm /% td /% ty»
- %td День месяца в формате двух цифр. например «01» до «31».
- %te День месяца, отформатированный без начального 0, например От «1» до «31».
- Дата в формате %tF ISO 8601 с «% tY-% tm-% td».
- %tH час дня для 24-часовых часов, например «00» — «23».
- %th То же, что %tb.
- %tI Час дня для 12-часовых часов, например «01» — «12».
- %tj День года, отформатированный с ведущими 0, например «001» — «366».
- %tk Час дня для 24-часовых часов без ведущего 0, например От «0» до «23».
- %tI Час дня для 12-часового клика без ведущего 0, например «1» до «12».
- %tM Минуты в течение часа, отформатированные в начале 0, например От «00» до «59».
- %tm Месяц отформатирован с начальным 0, например «01» — «12».
- %tN наносекунда, отформатированная с 9 цифрами и начальными 0, например. «000000000» — «999999999».
- %tp Специфичный для локали маркер «am» или «pm».
- %tQ Миллисекунды с 1 января 1970 г. 00:00:00 UTC.
- %tR Время отформатировано как 24 часа, например «% ЧТ:% ТМ».
- %tr Время отформатировано как 12 часов, например «% ТI:% tM:% tS% Тр».
- %tS секунды в течение минуты, отформатированные с 2 цифрами, например, От «00» до «60». «60» требуется для поддержки високосных секунд.
- %ts Секунды с 1 января 1970 г. 00:00:00 UTC.
- %tT Время отформатировано как 24 часа, например «% ЧТ:% Тm% Ts».
- %tY Год отформатирован 4 цифрами, например «0000» — «9999».
- %ty Год форматируется двумя цифрами, например От «00» до «99».
- %tZ Сокращение часового пояса. например «UTC», «PST» и т. Д.
- %T — Смещение часового пояса от GMT, например «-0800».

Аргумент задается как число, заканчивающееся на «\$» после «%», и выбирает указанный аргумент в списке аргументов.

```
1 String.format("%2$s", 32, "Hello"); // prints: "Hello"
```

Форматирование целых чисел

С помощью спецификатора формата %d можно использовать аргумент всех целочисленных типов, включая byte, short, int, long и BigInteger.

Форматирование по умолчанию:

```
1 String.format("%d", 93); // prints 93
```

Указание ширины:

```
1 String.format("|%20d|", 93); // prints: | 93|
```

Выравнивание по левому краю в пределах указанной ширины:

```
1 String.format("|%-20d|", 93); // prints: |93 |
```

Вывести положительные числа со знаком «+»:

(Отрицательные числа всегда включают «-»):

```
1 String.format("|%+20d|", 93); // prints: | +93|
```

Пробел перед положительными числами.

```
1 String.format("|% d|", 93); // prints: | 93| String.format("|% d|", -36); //  
1 prints: |-36|
```

Заключить отрицательные числа в круглые скобки («()») и пропустить «-»:

```
1 String.format("|%(d|", -36); // prints: |(36)|
```

Восьмеричный вывод:

```
1 String.format("|%o|", 93); // prints: 135
```

Шестнадцатеричный вывод:

```
1 String.format("|%x|", 93); // prints: 5d
```

Альтернативное представление для восьмеричного и шестнадцатеричного вывода:

Печатает восьмеричные числа с начальным «0» и шестнадцатеричные числа с начальным «0x».

```
1 String.format("|%#o|", 93);  
2 // prints: 0135  
3 String.format("|%#x|", 93);  
4 // prints: 0x5d  
5 String.format("|%#X|", 93);  
6 // prints: 0X5D
```

Форматирование строк и символов

Форматирование по умолчанию:

Печатает всю строку.

```
1 String.format("|%s|", "Hello World"); // prints: "Hello World"
```

С указанием длины поля:

```
1 String.format("|%30s|", "Hello World"); // prints: | Hello World|
```

Выравнивание текста по левому краю:

```
1 String.format("|%-30s|", "Hello World"); // prints: |Hello World |
```

Вывести максимальное количество символов:

```
1 String.format("|%.5s|", "Hello World"); // prints: |Hello|
```

Задать ширину поля и максимальное количество символов:

```
1 String.format("|%30.5s|", "Hello World"); | Hello|
```

Примеры

Следующий код форматирует строку, целое число и число с плавающей запятой с точностью до 2 чисел после десятичной точки (.2f):

```
1 String title = "Effective Java";
2 float price = 33.953f;
3 System.out.format("%s is a great book. It is sold at %.2f USD today.%n",
  title, price);
```

Вывод:

Effective Java is a great book. It is sold at 33.95 USD today.

В следующем примере используется флаг '-' для выравнивания по левому краю строк с шириной 30 символов:

```
1 List<string> listBook = Arrays.asList(
2     "Head First Java",
3     "Effective Java",
4     "The Passionate Programmer",
5     "Head First Design Patterns"
6 );
7
8 for (String book : listBook) {
9     System.out.format("%-30s - In Stock%n", book);
10 }
```

В следующем примере печатаются числа как в десятичном формате (%d), так и в шестнадцатеричном формате (%x and %#x):

```

1 System.out.format("Min value of a byte: %d (%1$#x)%n", Byte.MIN_VALUE);
2 System.out.format("Max value of a byte: %d (%1$#x)%n", Byte.MAX_VALUE);
3 System.out.format("Min value of an int: %d (%1$#x)%n", Integer.MIN_VALUE);
4 System.out.format("Max value of an int: %d (%1$#x)%n", Integer.MAX_VALUE);

```

Обратите внимание, что есть два спецификатора формата, но только один аргумент, поэтому мы используем индекс аргумента 1\$, чтобы указать позицию аргумента для спецификатора. Вывод:

```

Min value of a byte: -128 (0x80)
Max value of a byte: 127 (0x7f)
Min value of an int: -2147483648 (800000000)
Max value of an int: 2147483647 (7fffffff)

```

Следующий код использует префикс t с преобразованиями даты и времени (T, F и D) для записи отформатированных строк текущего времени и даты в текстовый файл:

```

1 PrintWriter writer = new PrintWriter(new File("datetime.txt"));
2
3 writer.format("Now is %tT %n", new Date());
4 writer.format("Today is %tF %n", new Date());
5 writer.format("Today is %tD %n", new Date());
6
7 writer.close();

```

На выходе получаем файл (datetime.txt), который будет иметь следующее содержимое:

```

Now is 23:25:11
Today is 2016-12-14
Today is 12/14/16

```

Следующий код форматирует объект Date в строку, используя полное локальное имя дня недели (tA), полное имя месяца (tB), день месяца (td) и год из 4 цифр (tY):

```

String longDate = String.format("Today is %tA %<tB %<td, %<tY", new Date());
System.out.println(longDate);

```

Метод String.format() возвращает отформатированную строку, укажите <, чтобы показать, что этот спецификатор использует тот же аргумент, что и предыдущий спецификатор.

Вывод: Today is Wednesday December 14, 2016

Теперь форматируем вывод, чтобы он выглядел как столбцы:

```

1 String specifiers = "%-30s %-20s %-5.2f%n";
2 System.out.format(specifiers, "Head First Java", "Kathy Sierra", 23.99f);
3 System.out.format(specifiers, "Thinking in Java", "Bruce Eckel", 25.69f);
4 System.out.format(specifiers, "Effective Java", "Joshua Bloch", 27.88f);
5 System.out.format(specifiers, "The Passionate Programmer", "Chad Fowler",
6 41.99f);
6 System.out.format(specifiers, "Code Complete", "Steve McConnell", 38.42f);

```

И самое сложное выполним форматирование чисел в Java с добавлением нуля для чисел менее 5 символов (%05d); заключим в скобки отрицательные числа (%(d) и включим знак для чисел (% + d):

```
1 System.out.format("%05d %n", 12);
2 System.out.format("%05d %n", 2016);
3 System.out.format("%05d %n", 365);
4 System.out.format("%05d %n", 19001800);
5 System.out.format("%(d %n", -1234);
6 System.out.format("%+d %n", 567);
7 System.out.format("%+d %n", -891);
```

Получим:

```
00012
02016
00365
19001800
(1234)
+567
-891
```