# 3ª Parte

## Formatando

Objetos de cadeias de caracteres que contenham a implementação de formatação são instâncias de PrintWriter, uma classe de cadeias de caracteres, e PrintStream, uma classe de controle de bytes.

Dois níveis de formatação estão disponíveis:

- ***print*** e ***println*** formatam valores individuais em um modo padrão;
- ***format*** formata qualquer valor numérico baseado numa String, quando são necessárias várias opções para uma formatação.

## Métodos print e println

Os métodos ***print*** ou ***println*** enviam para a saída um simples valor após convertê-lo usando um método ***toString*** apropriado. Podemos ver o que ocorre no exemplo Root:

```
public class Root {
   public static void main(String[] args) {
      int i = 2;
      double r = Math.sqrt(i);

      System.out.print("The square root of ");
      System.out.print(i);
      System.out.print(" is ");
      System.out.print(r);
      System.out.println(".");

      i = 5;
      r = Math.sqrt(i);
      System.out.println("The square root of " + i + " is " + r + ".");
   }
}
```

A saída de Root será:

```
The square root of 2 is 1.4142135623730951.
The square root of 5 is 2.23606797749979.
```

As variáveis i e r são formatadas duas vezes: na primeira vez utilizando um código que se sobrepõe ao ***print***, na segunda vez pela conversão automática gerada pelo compilador Java, que também se utiliza do método ***toString***. Você pode formatar qualquer valor dessa forma, porém não terá muito controle dos resultados.

## O Método format

O método ***format*** formata múltiplos argumentos baseado numa string de formatação. A string de formatação consiste em um texto estático que segue as especificações de uma formatação. A string formatada não sofre alterações no seu conteúdo, apenas o que será mostrado na saída padrão é que sairá formatado.

A formatação de strings possui várias opções. Cobriremos aqui algumas das mais básicas. Para uma completa descrição, consulte format string syntax nas especificações da API.

O exemplo Root2 formata dois valores com a simples chamada do método ***format***:

```
public class Root2 {
   public static void main(String[] args) {
      int i = 2;
      double r = Math.sqrt(i);

      System.out.format("The square root of %d is %f.%n", i, r);
   }
}
```

Aqui está a saída do Root2:

```
A raiz quadrada de 2 é 1.414214.
```

Assim como os três formatos usados neste exemplo, todas as especificações de formatação começam com um caractere % e terminam com 1 ou 2 caracteres de conversão que especificam o tipo de saída formatada que está sendo gerada. Os três tipos de conversores usados aqui são:

- d formata um valor inteiro para um valor decimal.
- f formata um valor de ponto flutuante (float) como um valor decimal.
- n coloca uma terminador de linha de acordo com a plataforma especificada.

Aqui estão algumas outras conversões:

x formata um inteiro para seu valor hexadecimal.

s formata qualquer valor para uma string.

tB formata um inteiro para um nome de mês na plataforma local.

Existem muitas outras conversões.

**Nota:** Exceto para %% e %n, todas as especificações de formatação devem seguir seus argumentos. Caso contrário acontecerá um erro de exceção.

Na linguagem de programação Java, o comando \n sempre gerará o caractere linefeed (pular linha) (\u000A). Não use \n, ao menos que você queira realmente um caractere de linefeed. Para poder utilizar o caractere correto de pular linha da plataforma local, utilize %n.

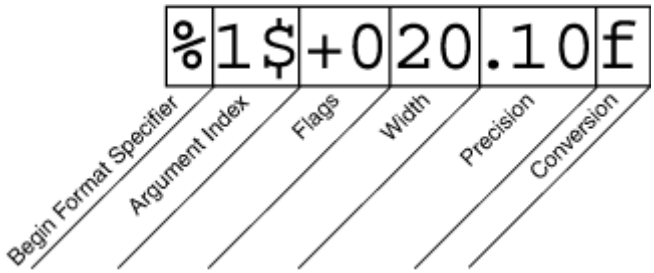
Somando-se as possibilidades de conversão, as especificações de formatos podem conter diversos elementos adicionais que permitirão customizar a saída formatada. Aqui está um exemplo, Format, que utilize todas as possibilidades desse tipo de elementos.

```
public class Format {
   public static void main(String[] args) {
      System.out.format("%f, %1$+020.10f %n", Math.PI);
   }
}
```

Eis a saída:

```
3.141593, +00000003.1415926536
```

Os elementos adicionais são opcionais. A seguinte figura mostra como é o posicionamento desses elementos.




Os elementos devem aparecer na mesma ordem mostrada acima. A partir da direita, os elementos opcionais são:

- **Precision**: para valores de ponto flutuante, este é o elemento de precisão do valor formatado. Para “s” e outros tipos comuns de conversão, este é o tamanho máximo de valores formatados; o valor será truncado à direita se necessário.

- **Width**: o tamanho mínimo de um valor formatado; o valor será “acomodado” se necessário. Por padrão o valor é preenchido à esquerda com espaços.

- **Flags**: especifica opções adicionais de formatação. No exemplo Format, a flag “+” flag especifica que o número deverá sempre ser formatado com

um sinal, e a flag “0” flag especifica que o caractere “0” será o caractere de preenchimento. Outras flags incluem - (ajuste à direita) e, (formata um número com a especificação da plataforma local para separadores de milhares). Note que algumas flags não podem ser usadas com algumas outras flags ou com certos tipos de conversões.

- O **Argument Index** permite explicitar o casamento com um argumento designado. Você também poderá especificar “<” para casar o mesmo argumento com sua prévia especificação. Assim o exemplo ficaria dessa forma:

```
System.out.format("%f, %<+020.10f %n", Math.PI);
```