03

Trabalhando com saída no console

Imprimindo no console padrão

Para imprimir na saída do console, podemos usar o método `print*` do objeto `out`, da classe `System`:

```
System.out.print("hello world");
```

COPIAR CÓDIGO

System.out é um PrintStream

O `out` é um atributo estático e público da classe `System`, do tipo `PrintStream`. A classe `PrintStream` possui vários métodos que permitem escrever diversos tipos de dados de maneira padronizada em um `OutputStream`.

O método `print` possui várias sobrecargas, recebendo desde ::strings:: até tipos primitivos:

```
System.out.print(false); // a boolean
System.out.print(10.3);  // a double value
System.out.print("Some text"); // some text
```

COPIAR CÓDIGO

Também existe uma versão de `print` que recebe `Object`, ou seja, qualquer objeto. Neste caso, será invocado o método `toString` do objeto passado:

```
class Test {

    public String toString(){
        return "My test object...";
    }

    public static void main(String[] args){
        System.out.print(new Test()); // My test object...
    }
}
```

COPIAR CÓDIGO

Existe um único caso especial que foge à regra; além de primitivos, ::strings:: e `Object`, existe também uma sobrecarga que recebe um array de `char`. Todos os outros ::arrays:: são tratados como `Object`

```
char[] c = {'a','b','c'};
int[] i = {1,2,3};
System.out.print(c); // abc
System.out.print(i); // [I@9d8643e (ou similar)]
```

COPIAR CÓDIGO

O método `print` apenas imprime o valor passado. Se invocado várias vezes em sequência, todos os valores serão impressos em uma única linha:

```
System.out.print(false);
System.out.print(10.3);
System.out.print("Some text");
```

COPIAR CÓDIGO

Esse código imprime:

```
false10.3Some text
```

COPIAR CÓDIGO

Podemos imprimir cada conteúdo em uma linha diferente, concatenando um `"\n"` após cada impressão. Para este caso, existe o método `println`, que adiciona uma quebra de linha após cada chamada:

```
System.out.println(false);
System.out.println(10.3);
System.out.println("Some text");
```

COPIAR CÓDIGO

Já esse código imprime:

```
false
10.3
Some text
```

COPIAR CÓDIGO

O `println` possui as mesmas sobrecargas que o método `print`, além de uma versão sem nenhum parâmetro, que apenas quebra uma linha:

```
System.out.print("foo");
System.out.println(); // line break
System.out.print("bar");
```

COPIAR CÓDIGO

Imprime

```
foo
bar
```

COPIAR CÓDIGO

Formatando a impressão

Na versão 5 do Java, foram incluídos dois métodos para permitir a impressão no console de modo formatado, `format` e `printf`. Ambos se comportam exatamente da mesma maneira, sendo que o `printf` foi incluído provavelmente apenas para manter uma sintaxe próxima à da linguagem `C`, na qual existe um método com comportamento similar com este nome. Sendo assim, tudo o que discutirmos sobre um método serve para o outro.

O `printf` recebe dois parâmetros: o primeiro é uma `String` que pode conter apenas texto normal ou incluir caracteres especiais de formatação; o segundo é um `varargs` de objetos a serem usados na impressão. Vamos começar com um exemplo simples:

```
System.out.printf("Hello %s, have a nice day!", "Mario");
```

COPIAR CÓDIGO

Esse código imprime:

```
Hello Mario, have a nice day!
```

COPIAR CÓDIGO

Perceba o caractere `%` na mensagem. É ele que indica que ali temos uma formatação especial. O `s` logo em seguida ao `%` indica que lá incluiremos alguma outra `String`, que é a que passamos como segundo argumento deste método. O próprio método se encarrega de fazer a concatenação no lugar certo e exibir o resultado no console, o que chamamos de interpolação de `String` (::string interpolation::).

Para indicar como a formatação deve ser feita, usamos a seguinte estrutura, que veremos com mais detalhes a partir de agora:

```
%[index$][flags][width][.precision]type
```

COPIAR CÓDIGO

Todas as opções entre **[ ]** são opcionais. Somos obrigados a informar apenas o caractere de **%** e o tipo do argumento que será concatenado. Vamos analisar cada opção separadamente:

type é o tipo de argumento que será passado e suporta os seguintes valores:

- `b` - `boolean`
- `c` - `char`
- `d` - Números inteiros

- `f` - Números decimais
- `s` - `String`
- `n` - Quebra de linha

Vejamos alguns exemplos:

```
System.out.printf("%s %n", "foo"); //foo
System.out.printf("%b %n", false); //false
System.out.printf("%d %n", 42);    //42
System.out.printf("%d %n", 1024L); //1024
System.out.printf("%f %n", 23.9f); //23.900000
System.out.printf("%f %n", 44.0);  //44.000000
```

COPIAR CÓDIGO

Repare que podemos passar mais de uma instrução por impressão, sendo que cada `%` está relacionado com o próximo parâmetro passado na sequência:

```
System.out.printf("%s, it's %b, the result is %d", "yes",
                  true, 100);
```

COPIAR CÓDIGO

Esse código imprime:

```
yes, it's true, the result is 100
```

COPIAR CÓDIGO

O **index** é um número inteiro delimitado pelo caractere `$`, que indica qual dos argumentos deve ser impresso nessa posição se desejarmos fugir do padrão

sequencial. Por exemplo:

```
System.out.printf("%2$s %1$s", "World","Hello"); // Hello
World
```

COPIAR CÓDIGO

Mesmo passando os argumentos fora de ordem, conseguimos indicar na ::string:: qual a ordem que queremos na impressão. Esse recurso é extremamente interessante quando queremos repetir o mesmo valor em dois pontos da nossa formatação.

width indica a quantidade mínima de caracteres para imprimir. Completa com espaços à esquerda caso o valor seja menor que a largura mínima. Caso seja maior, não faz nada:

```
System.out.printf("[%5d]%n", 22);       //[   22]
System.out.printf("[%5s]%n", "foo");    //[  foo]
System.out.printf("[%5s]%n", "foofoo"); //[foofoo]
```

COPIAR CÓDIGO

flags são caracteres especiais que alteram a maneira como a impressão é feita. Para a prova, é importante conhecer alguns, dentre os quais os dois que indicam se o número é positivo ou negativo:

- `+` - Sempre inclui um sinal de positivo (+) ou negativo (-) em números.
- `(` - Números negativos são exibidos entre parênteses.

Dois de alinhamento à esquerda ou direita:

- `-` - Alinha à esquerda. Precisa de tamanho para ser usado.
- `0` - Completa a esquerda com zeros. Precisa de tamanho para ser usado.

Juntamente com o tamanho mínimo, podemos usar a ::flag:: de alinhamento e de completar com zeros:

```
System.out.printf("[%05d]%n", 22);    //[00022]
System.out.printf("[%-5s]%n", "foo"); //[foo  ]
```

COPIAR CÓDIGO

Só é possível completar com zeros quando estamos formatando números. Tentar usar esta flag com ::strings:: lança uma `FormatFlagsConversionMismatchException`, como em:

```
System.out.printf("[%05s]%n", "foo");
```

COPIAR CÓDIGO

Temos uma flag para separar casa de milhares e decimais:

- `,` - Habilita separadores de milhar e decimal.

Vamos ver alguns exemplo:

```
System.out.printf("%+d %n", 22);        //+22
System.out.printf("%,f %n", 1234.56);  //1,234.560000
System.out.printf("%(f %n", -1234.56); //(1234.560000)
```

COPIAR CÓDIGO

A impressão de separadores de milhar e decimal depende da linguagem do sistema onde o código é executado. Podemos alterar o padrão usando um objeto da classe `Locale`, que altera as regras padrões para as regras da língua informada:

```
Locale br = new Locale("pt","BR");
System.out.printf(br,"%,f %n", 123456.789); // 123.456,789000
```

COPIAR CÓDIGO

Quando da formatação de números com casas decimais, **precision** indica quantas casas queremos depois da vírgula, basta usar um `.` seguido do número de caracteres. Vale lembrar que só é possível mudar a precisão quando estamos formatando números decimais.

```
System.out.printf("[%.2f]%n", 22.5); //[22.50]
```

COPIAR CÓDIGO