Conteúdo do Capítulo

2.1 Tipos e variáveis

Em Java, cada valor é de um tipo.

Em Java, cada valor é de um tipo. Por exemplo, `"Hello, World"` é do tipo `String`, o objeto `System.out` é do tipo `PrintStream` e o número 13 é do tipo `int` (uma abreviação para "integer", ou inteiro). O tipo informa o que você pode fazer com os valores. Você pode chamar `println` em qualquer objeto do tipo `PrintStream`. Pode também calcular a soma ou o produto de dois inteiros quaisquer.

É muito comum querermos armazenar os valores para utilizá-los posteriormente. Para lembrar-se de um objeto, você precisa armazená-lo em uma *variável*. Uma variável é um local de armazenamento na memória do computador que possui um *tipo*, um *nome* e um conteúdo. Por exemplo, aqui declaramos três variáveis:

```
String greeting = "Hello, World!";
PrintStream printer = System.out;
int luckyNumber = 13;
```

Usa-se variáveis para armazenar valores que se deseja utilizar em um momento posterior.

A primeira variável chama-se `greeting`. Ela pode ser utilizada para armazenar valores do tipo `String` e é configurada com o valor `"Hello, World!"`. A segunda variável armazena um valor do tipo `PrintStream` e a terceira armazena um inteiro.

Variáveis podem ser utilizadas no lugar dos objetos que elas armazenam:

```
printer.println(greeting); // O mesmo que System.out.println("Hello, World!")
printer.println(luckyNumber); // O mesmo que System.out.println(13)
```

SINTAXE 2.1 Definição de variável

nomeDoTipo nomeDaVariável = valor;
ou
nomeDoTipo nomeDaVariável;

Exemplo:

```
String greeting = "Hello, Dave!";
```

Objetivo:

Definir uma nova variável de um tipo particular e, opcionalmente, fornecer um valor inicial

Ao declarar suas próprias variáveis, você precisa tomar duas decisões.

- Qual tipo você deve utilizar para a variável?
- Qual nome você deve atribuir à variável?

O tipo depende do uso final. Se precisar armazenar uma string, utilize o tipo `String` para sua variável.

É um erro armazenar um valor cuja classe não corresponde ao tipo da variável. Por exemplo, o seguinte é um erro:

```
String greeting = 13; // ERRO: Tipos incompatíveis
```

Você não pode utilizar uma variável `String` para armazenar um inteiro. O compilador verifica não-correspondências de tipo para protegê-lo contra erros.

Identificadores para variáveis, métodos e classes são compostos de letras, dígitos e caracteres de sublinhado.

Ao decidir sobre um nome para uma variável, você deve fazer uma escolha que descreve o propósito da variável. Por exemplo, o nome da variável `greeting` é uma escolha melhor que o nome `g`.

Um *identificador* é o nome de uma variável, método ou classe. Java impõe as seguintes regras para identificadores:

- Identificadores podem ser compostos de letras, dígitos, caracteres de sublinhado (_) e sinal de cifrão ($). Eles, porém, não podem iniciar com um dígito. Por exemplo, `greeting1` é válido, mas `1greeting` não.
- Você não pode utilizar outros símbolos como `?` ou `%`. Por exemplo, `hello!` não é um identificador válido.
- Não são permitidos espaços em identificadores. Portanto, `lucky number` não é válido.
- Além disso, você não pode utilizar *palavras reservadas*, como `public`, como nomes; essas palavras são reservadas exclusivamente para seus significados especiais em Java.
- Identificadores também fazem *distinção entre letras maiúsculas e minúsculas*; isto é, `greeting` e `Greeting` são *diferentes*.

Por convenção, nomes de variáveis devem iniciar com uma letra minúscula.

Essas são regras rígidas da linguagem Java. Se violar uma delas, o compilador informará um erro. Além disso, há algumas *convenções* que você deve obedecer para que seus programas possam ser lidos facilmente por outros programadores:

- Nomes de variáveis e métodos devem iniciar com letra minúscula. É válido utilizar uma letra maiúscula ocasionalmente, como `luckyNumber`. Essa combinação de letras minúsculas e maiúsculas às vezes é chamada de "notação camelo" porque as letras maiúsculas se destacam como a corcova de um camelo.
- Nomes de classes devem iniciar com letra maiúscula. Por exemplo, `Greeting` seria um nome apropriado para uma classe, mas não para uma variável.

Se violar essas convenções, o compilador não reclamará, mas você irá confundir outros programadores que lêem seu código.

Autoverificação da aprendizagem

1. Qual é o tipo dos valores `0` e `"0"`?
2. Quais dos seguintes são identificadores válidos?

```
Greeting1
g
void
101dalmatians
Hello, World
<greeting>
```

3. Defina uma variável para armazenar seu nome. Utilize a notação camelo no nome da variável.

2.2 O operador de atribuição

Utilize o operador de atribuição (=) para alterar o valor de uma variável.

Você pode alterar o valor de uma variável existente com o operador de atribuição (=). Por exemplo, considere a definição da variável a seguir:

```
int luckyNumber = 13; ❶
```

Se quiser alterar o valor dessa variável, simplesmente atribua o novo valor:

```
luckyNumber = 12; ❷
```

A atribuição substitui o valor original da variável (veja Figura 1).

Na linguagem de programação Java, o operador = denota uma *ação*, substituir o valor de uma variável. Esse uso difere do uso tradicional do símbolo =, como um operador de igualdade.

É um erro utilizar uma variável à qual nunca foi atribuído um valor. Por exemplo, a seqüência de instruções

```
int luckyNumber;
System.out.println(luckyNumber);    // ERRO – variável não-inicializada
```

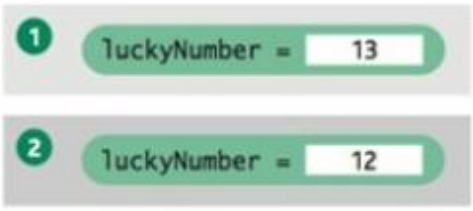
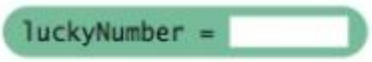


Figura 1
Atribuindo um novo valor a uma variável.

Figura 2
Uma variável objeto não-inicializada.



é um erro. O compilador reclama sobre uma "variável não-inicializada" quando você usa uma variável à qual nunca foi atribuído valor algum. (Veja Figura 2.)

Todas as variáveis devem ser inicializadas antes de você acessá-las.

O correto é atribuir um valor à variável antes de utilizá-la:

```
int luckyNumber;
luckyNumber = 13;
System.out.println(luckyNumber); // OK
```

Ou, melhor ainda, inicialize a variável ao defini-la.

```
int luckyNumber = 13;
System.out.println(luckyNumber); // OK
```

SINTAXE 2.2 Atribuição

nomeDaVariável = *valor*;

Exemplo:

```
luckyNumber = 12;
```

Objetivo:

Atribuir um novo valor a uma variável previamente definida

AUTOVERIFICAÇÃO DA APRENDIZAGEM

4. `12 = 12` é uma expressão válida na linguagem Java?
5. Como você altera o valor da variável `greeting` para `"Hello, Nina!"`?

2.3 Objetos, classes e métodos

Objetos são entidades no seu programa que você manipula invocando métodos.

Um *objeto* é uma entidade que você pode manipular no seu programa. Normalmente, você não sabe como o objeto é organizado internamente, mas ele tem um comportamento bem-definido e é isso o que nos importa quando o utilizamos.

Um método é uma seqüência de instruções que acessam os dados de um objeto.

Você manipula um objeto chamando um ou mais dos seus *métodos*. Um método consiste em uma seqüência de instruções que acessam os dados internos. Quando você chama o método, não há como saber exatamente quais são essas instruções, mas você sabe o propósito do método.

Por exemplo, vimos no Capítulo 1 que `System.out` refere-se a um objeto. Você o manipula chamando o método `println`. Quando o método `println` é chamado, algumas atividades ocorrem dentro do objeto e o efeito final é que o texto aparece na janela da console. Você não sabe como isso acontece, mas isso é válido. O importante é que o método execute aquilo que você solicitou.

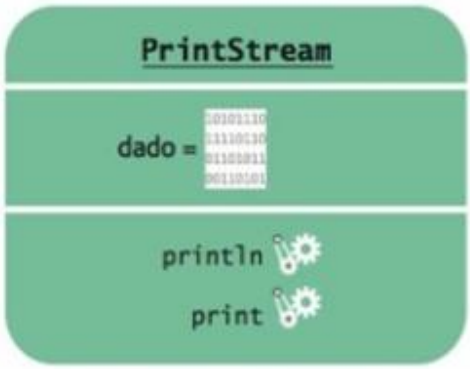


Figura 3
Representação do objeto `System.out`.

A Figura 3 mostra uma representação do objeto `System.out`. Os dados internos são simbolizados por uma seqüência de zeros e uns. Pense em cada método (simbolizado pelas engrenagens) como uma parte de maquinaria que executa uma determinada tarefa.

No Capítulo 1, você viu dois objetos:

- `System.out`
- `"Hello, World!"`

Esses objetos pertencem a diferentes *classes*. O objeto `System.out` pertence à classe `PrintStream`. O objeto `"Hello, World!"` pertence à classe `String`. Uma classe especifica os métodos que você pode aplicar aos objetos dela.

Uma classe define os métodos que você pode aplicar aos objetos dela.

Você pode utilizar o método `println` com qualquer objeto pertencente à classe `PrintStream`. `System.out` é um desses objetos. É possível obter outros objetos da classe `PrintStream`. Por exemplo, você pode construir um objeto `PrintStream` com o objetivo de enviar a saída para um arquivo. Mas só discutiremos arquivos no Capítulo 11.

Assim como a classe `PrintStream` fornece métodos como `println` e `print` para seus objetos, a classe `String` fornece métodos que você pode aplicar a objetos `String`. Um deles é o método `length`. O método `length` conta o número de caracteres em uma string. Você pode aplicar esse método a qualquer objeto do tipo `String`. Por exemplo, a seqüência de instruções

```
String greeting = "Hello, World!";
int n = greeting.length();
```

inicializa `n` com o número de caracteres do objeto `String` `"Hello, World!"`. Depois de as instruções no método `length` serem executadas, `n` é configurado como 13. (As aspas não são parte da string e o método `length` não as conta.)

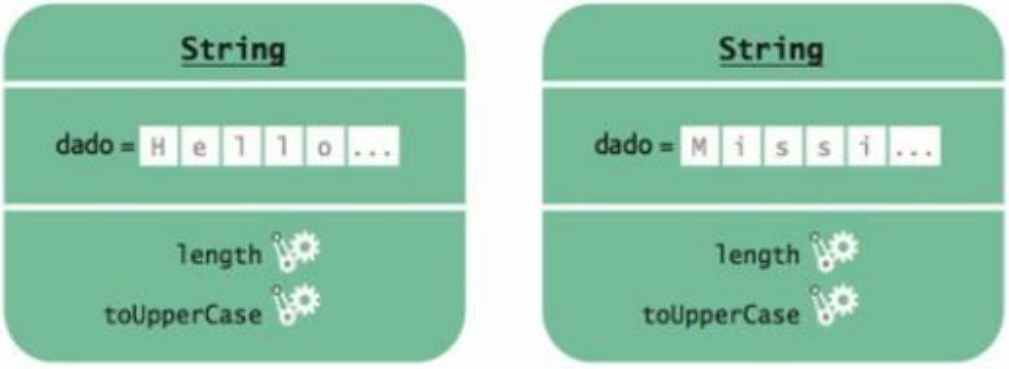


Figura 4 Uma representação de dois objetos `String`.

O método `length` – diferentemente do método `println` – não requer entrada dentro dos parênteses. Entretanto, o método `length` fornece uma saída, a saber, a contagem de caracteres.

Na próxima seção, você verá mais detalhadamente como fornecer entradas a um método e obter saídas do método.

Vejamos outro método da classe `String`. Quando você aplica o método `toUpperCase` a um objeto `String`, esse método cria outro objeto `String` que contém os caracteres da string original, com as letras minúsculas convertidas em maiúsculas. Por exemplo, a seqüência de instruções

```
String river = "Mississippi";
String bigRiver = river.toUpperCase();
```

configura `bigRiver` como o objeto `String` `"MISSISSIPPI"`.

Ao aplicar um método a um objeto, você deve certificar-se de que esse método esteja definido na classe apropriada. Por exemplo, é um erro chamar

```
System.out.length(); // Essa chamada de método é um erro.
```

A classe `PrintStream` (à qual `System.out` pertence) não possui um método `length`.

A interface pública de uma classe especifica o que você pode fazer com os objetos dela. A implementação oculta descreve como essas ações são executadas.

Vamos resumir. Em Java, *cada objeto pertence a uma classe. A classe define os métodos para os objetos.* Por exemplo, a classe `String` define os métodos `length` e `toUpperCase` (bem como outros métodos – veremos a maioria deles no Capítulo 4). Os métodos formam a interface *pública* da classe e determinam o que você pode fazer com os objetos dela. Uma classe também define uma *implementação privada*, que descreve os dados dentro dos seus objetos e as instruções para seus métodos. Esses detalhes permanecem ocultos dos programadores, os quais utilizam objetos e métodos de chamada.

A Figura 4 mostra dois objetos da classe `String`. Cada objeto armazena seus próprios dados (desenhados como caixas que contêm caracteres). Ambos suportam o mesmo conjunto de métodos – a interface que é especificada pela classe `String`.

AUTOVERIFICAÇÃO DA APRENDIZAGEM

6. Como você pode calcular o comprimento da string `"Mississippi"`?
7. Como você pode imprimir a versão em letras maiúsculas de `"Hello, World!"`?
8. É válido chamar `river.println()`? Por que sim ou por que não?

2.4 Parâmetros de método e valores de retorno

Nesta seção, examinaremos como fornecer entradas em um método e como obter a saída do método.

Um parâmetro é uma entrada para um método.

Alguns métodos requerem entradas que fornecem detalhes sobre o trabalho que precisam fazer. Por exemplo, o método `println` tem uma entrada: a string que deve ser impressa. Cientistas da computação utilizam o termo técnico *parâmetro* para entradas de método. Dizemos que a string `greeting` é um parâmetro da chamada de método

```
System.out.println(greeting)
```

A Figura 5 ilustra a passagem do parâmetro para o método.

O parâmetro implícito de uma chamada de método é o objeto em que o método é invocado.

Tecnicamente falando, o parâmetro `greeting` é um *parâmetro explícito* do método `println`. O objeto em que você invoca o método também é considerado um parâmetro da chamada de método, e é denominado *parâmetro implícito*. Por exemplo, `System.out` é o parâmetro implícito da seguinte chamada de método:

```
System.out.println(greeting)
```

Alguns métodos requerem múltiplos parâmetros explícitos, outros não requerem absolutamente nenhum. Um exemplo do último é o método `length` da classe `String` (veja Figura 6). Todas as informações que o método `length` requer para fazer o trabalho – a saber, a seqüência de caracteres da string – estão armazenadas no próprio objeto parâmetro implícito.

O valor de retorno de um método é o resultado que o método calculou para uso pelo código que o chamou.

O método `length` difere do método `println` de uma outra maneira: ele tem uma saída. Dizemos que o método *retorna um valor*, a saber, o número de caracteres na string. Você pode armazenar o valor de retorno em uma variável:

```
int n = greeting.length();
```

Você também pode utilizar o valor de retorno como um parâmetro de outro método:

```
System.out.println(greeting.length());
```

A chamada de método `greeting.length()` retorna um valor – o inteiro 13. O valor de retorno torna-se um parâmetro do método `println`. A Figura 7 mostra o processo.

Nem todos os métodos retornam valores. Um exemplo é o método `println`. O método `println` interage com o sistema operacional, fazendo com que os caracteres apareçam em uma janela. Mas ele não retorna um valor ao código que o chama.

Vamos analisar uma chamada de método mais complexa. Aqui, chamaremos o método `replace` da classe `String`. O método `replace` executa uma operação de pesquisa e substituição, semelhante àquela de um processador de texto. Por exemplo, a chamada:

```
river.replace("issipp", "our")
```

constrói uma nova string que é obtida substituindo todas as ocorrências de `"issipp"` em `"Mississippi"` por `"our"`. (Nessa situação, há somente uma substituição.) O método retorna o objeto `String` `"Missouri"` (que tanto pode ser salvo em uma variável como passado para outro método).

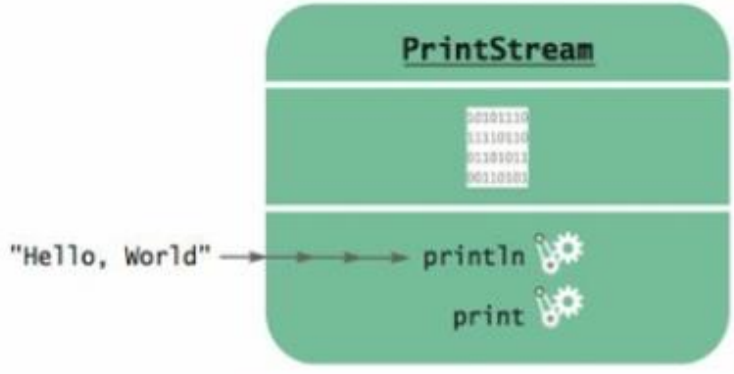


Figura 5 Passando um parâmetro para o método `println`.

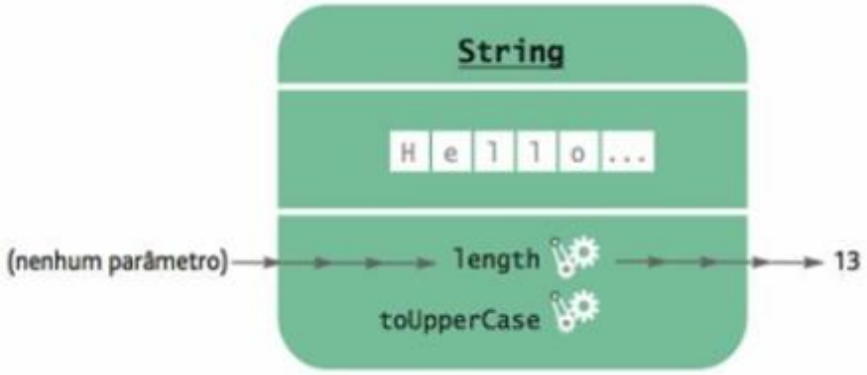


Figura 6 Invocando o método `length` em um objeto `String`.

Como a Figura 8 mostra, essa chamada de método tem:

- um parâmetro implícito: a string `"Mississippi"`
- dois parâmetros explícitos: as strings `"issipp"` e `"our"`
- um valor de retorno: a string `"Missouri"`

Quando um método é definido em uma classe, essa definição especifica os tipos dos parâmetros explícitos e o valor de retorno. Por exemplo, a classe `String` define o método `length` como:

```
public int length()
```

Isto é, não há parâmetro explícito e o valor de retorno é do tipo `int`. (Por enquanto, todos os métodos que consideramos serão métodos "públicos" – ver o Capítulo 10 para métodos mais restritos.)

O tipo do parâmetro implícito é a classe que define o método – `String` no nosso caso. Ele não é mencionado na definição de método – daí o termo "implícito".

O método `replace` é definido como

```
public String replace(String target, String replacement)
```

Para chamar o método `replace`, você fornece dois parâmetros explícitos, `target` e `replacement`, que são do tipo `String`. O valor retornado é uma outra string.

Quando um método não retorna valor algum, o tipo de retorno é declarado com a palavra reservada `void`. Por exemplo, a classe `PrintStream` define o método `println` como:

```
public void println(String output)
```

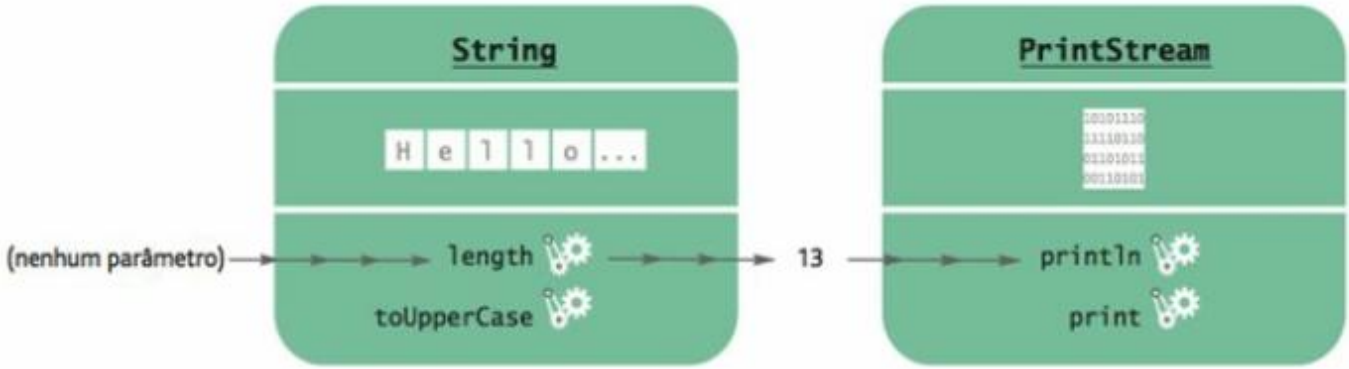


Figura 7 Passando o resultado de uma chamada de método para outro método.

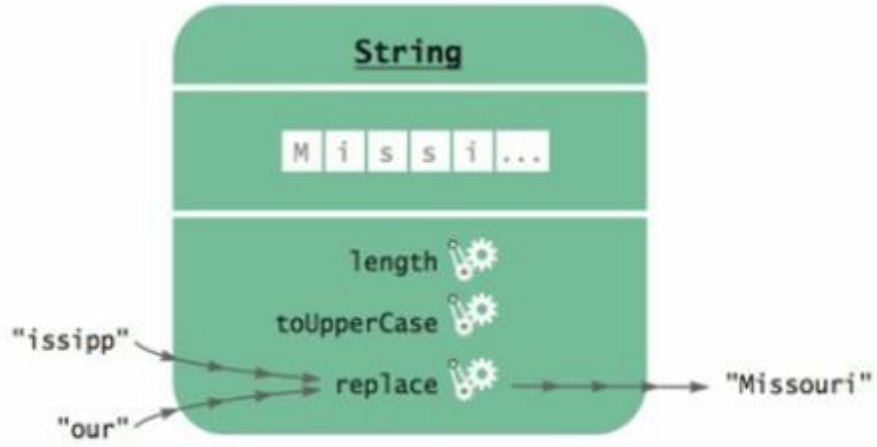


Figura 8 Chamando o método `replace`.

Um nome de método é sobrecarregado se uma classe tiver mais de um método com o mesmo nome (mas tipos diferentes de parâmetros).

Ocasionalmente, uma classe define dois métodos com o mesmo nome e diferentes tipos de parâmetros explícitos. Por exemplo, a classe `PrintStream` define um segundo método, também chamado `println`, como

```
public void println(int output)
```

Esse método é utilizado para imprimir um valor inteiro. Dizemos que o nome `println` é *sobrecarregado* porque referencia mais de um método.

Autoverificação da aprendizagem

9. Quais são os parâmetros implícitos, os parâmetros explícitos e os valores de retorno na chamada do método `river.length()`?
10. Qual é o resultado da chamada `river.replace("p", "s")`?
11. Qual é o resultado da chamada `greeting.replace("World", "Dave").length()`?
12. Como o método `toUpperCase` é definido na classe `String`?