Programação Estruturada

Aula 07 - Manipulação de Strings

Apresentação

Na aula passada, você aprendeu a programar em Java usando vetores e matrizes. Na aula de hoje, você vai aprender com mais detalhes como funciona a manipulação de Strings. Como já vimos nas aulas passadas, uma String é um texto, ou seja, uma sequência de caracteres. Existem diversas funções na linguagem Java para manipular Strings, e esse é justamente o tema desta nossa aula.

Faça uma boa leitura!

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

- Criar programas que utilizem os recursos de manipulação Strings.
- Manipular os caracteres que formam uma String.

1. Comparação de Strings

A primeira manipulação que você aprenderá nesta aula é a comparação entre duas Strings. Imagine um programa que lê do usuário dois nomes e que quer comparar se eles são iguais. Como você faria para comparar os dois nomes digitados? Se você está pensando em usar o operador de igualdade do Java (==), essa não é a forma mais recomendada. Isso porque o operador == deve ser utilizado com tipos primitivos, não com objetos. Você estudará em detalhes o conceito de objetos no módulo de Programação Orientada a Objetos, mas posso lhe adiantar que a comparação de objetos (neste caso, Strings) deve ser feita utilizando-se uma função chamada de **equals**.

Observe o programa a seguir:

```
1  import java.util.Scanner;
2
3  public class ProgramaEquals {
4      public static void main(String[] args) {
5          Scanner leitor = new Scanner(System.in);
6          System.out.println("Digite o primeiro nome:");
7          String nome1 = leitor.next();
8          System.out.println("Digite o segundo nome:");
9          String nome2 = leitor.next();
10         if (nome1.equals(nome2)) {
11             System.out.println("Nomes iguais");
12         } else {
13             System.out.println("Nomes diferentes");
14         }
15     }
16 }
```

O programa apresentado faz uso da expressão nome1.**equals**(nome2). A maioria das rotinas que vamos estudar na aula de hoje devem ser chamadas colocando o nome da variável que contém a String antes do nome da rotina e separado por um ponto. No caso do ProgramaEquals, usamos a variável nome1 para isso. A função equals recebe como parâmetro uma segunda String, que é justamente com a qual a primeira String (nome1) será comparada. Dessa forma, a função equals irá comparar os textos das variáveis nome1 e nome2, retornando *true*, caso elas sejam iguais, e falso caso contrário.

Vídeo 02 - Compara strings

Uma ressalva é importante. Em Java, a comparação de Strings é sensível a letras maiúsculas e minúsculas. Por exemplo, se você usar o ProgramaEquals para comparar os textos "Nomes iguais" e "nomes iguais", o programa irá indicar que os textos digitados são diferentes! Tudo isso apenas porque na primeira String a letra N está maiúscula e na segunda String a letra n está minúscula.

Atividade 01

1. Implemente e execute o ProgramaEquals, conforme mostrado na aula. Verifique o resultado ao digitar textos iguais e textos diferentes.
2. Comente o que acontece na execução do ProgramaEquals quando você passa dois textos com mesmas letras, porém com alguma diferença de letras entre maiúscula e minúscula.

Quando você quer comparar Strings considerando que não há distinção entre letras minúsculas e maiúsculas, você tem algumas opções. A primeira delas é converter os dois textos para letras maiúsculas antes de compará-los. Veja como isso é feito através do código a seguir.

```java
import java.util.Scanner;

public class ProgramaEqualsUpperCase {
    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);
        System.out.println("Digite o primeiro nome:");
        String nome1 = leitor.next();
        System.out.println("Digite o segundo nome:");
        String nome2 = leitor.next();
        nome1 = nome1.toUpperCase();
        nome2 = nome2.toUpperCase();
        if (nome1.equals(nome2)) {
            System.out.println("Nomes iguais");
        } else {
            System.out.println("Nomes diferentes");
        }
        System.out.println(nome1);
        System.out.println(nome2);
    }
}
```

Note o uso da função **toUpperCase()**. Ela também deve ser aplicada após o nome de uma variável do tipo String, devidamente separada por ponto. A função retorna o mesmo texto da variável String indicada antes do ponto, porém com todas as suas letras convertidas para maiúscula. Fazendo a conversão dos dois textos para maiúscula, podemos usar a função equals sem nos preocuparmos com diferenças entre maiúsculas ou minúsculas no texto. Também poderíamos ter transformado as duas Strings para minúsculas ao invés de maiúsculas. Para isso, faríamos uso da função **toLowerCase()**, cujo funcionamento é o inverso do toUpperCase().

Atividade 02

---

1. Implemente e execute o ProgramaEqualsUpperCase, conforme mostrado na aula. Verifique o resultado ao digitar os mesmos textos, variando apenas a questão de letras maiúsculas e minúsculas.

2. Altere a implementação do programa para transformar os dois textos para letras minúsculas. Verifique se o programa continua executando corretamente.

Apesar do ProgramaEqualsUpperCase funcionar, ele não mostra a solução mais indicada, porque os textos originais foram alterados. Observe que os nomes impressos no final do ProgramaEqualsUpperCase estão todos em maiúsculos. Para evitar esse problema, você pode fazer uso da rotina **equalsIgnoreCase**. Seu funcionamento e uso são praticamente iguais aos da função equals, a única coisa que muda é que na comparação não se faz distinção entre letras maiúsculas e minúsculas. Veja a seguir como fica o ProgramaEquals quando usado o equalsIgnoreCase:

```
1  import java.util.Scanner;
2
3  public class ProgramaEqualsIgnoreCase {
4     public static void main(String[] args) {
5        Scanner leitor = new Scanner(System.in);
6        System.out.println("Digite o primeiro nome:");
7        String nome1 = leitor.next();
8        System.out.println("Digite o segundo nome:");
9        String nome2 = leitor.next();
10       if (nome1.equalsIgnoreCase(nome2)) {
11          System.out.println("Nomes iguais");
12       } else {
13          System.out.println("Nomes diferentes");
14       }
15    }
16 }
```

Vídeo 03 - Compara Strings Ignore Case

Atividade 03

1. Implemente e execute o ProgramaEqualsIgnoreCase, conforme mostrado na aula. Verifique o resultado ao digitar os mesmos textos, variando apenas a questão de letras maiúsculas e minúsculas.

2. Strings como arrays de caracteres

Se você prestar atenção, irá perceber que uma String pode ser vista como uma sequência de caracteres, ou seja, um array de caracteres. De fato, você pode transformar uma String em um array de caracteres e vice-versa. Veja isso no programa a seguir, que imprime um texto de forma inversa:

```
1  import java.util.Scanner;
2
3  public class ProgramaStringCaracteres {
4      public static void main(String[] args) {
5          Scanner leitor = new Scanner(System.in);
6          System.out.println("Digite uma palavra:");
7          String nome1 = leitor.next();
8          char[] caracteres = nome1.toCharArray();
9          System.out.println("Inverso:");
10         for (int i = caracteres.length - 1; i >= 0 ; i--) {
11             System.out.print(caracteres[i]);
12         }
13     }
14 }
```

Note o uso da função **toCharArray()**. Essa função é aplicada a uma variável do tipo String, que no caso é nome1, e seu retorno é um array de caracteres com o tamanho exato do texto e que armazena exatamente os caracteres que formam o texto original. Essa conversão de String para array, porém, não é muito comum, pois existem funções que trabalham diretamente com Strings e que simulam esse comportamento de um array (indexação das letras, etc.). Vejamos como podemos construir esse mesmo programa apenas usando String:

```java
import java.util.Scanner;

public class ProgramaString {
    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);
        System.out.println("Digite uma palavra:");
        String nome1 = leitor.next();
        System.out.println("Inverso:");
        for (int i = nome1.length() - 1; i >= 0 ; i--) {
            System.out.print(nome1.charAt(i));
        }
    }
}
```

Observe que nessa nova versão do programa foram utilizadas as funções **length()** e **charAt()**. A primeira dessas funções retorna o tamanho da String, ou seja, a quantidade de caracteres que ela possui. E nessa contagem, espaços em branco e caracteres especiais como o \n também contam! Já a segunda função, recebe como parâmetro um número inteiro e retorna o caractere indexado pelo número indicado. Isso quer dizer que se você passar o número 0 (zero) como parâmetro, a função charAt() irá retornar o primeiro caractere da String. Lembre-se que os índices começam do número zero!

Essas duas funções permitem que você faça a leitura dos caracteres de uma String como se você estivesse trabalhando com um array de caracteres. Não é legal? Não há necessidade nesse caso de fazer nenhuma conversão.

Atividade 04

---

1. Implemente e execute os ProgramaStringCaracteres e ProgramaString, conforme mostrado na aula. Verifique se eles geram o mesmo resultado para entradas iguais.

2. Crie um programa que lê uma String e imprima apenas os caracteres de índice ímpar.

3. Buscas em Strings

Muitas vezes queremos verificar se um determinado texto contém uma palavra, ou se inicia ou termina em outra, etc. Além disso, podemos querer saber a posição não de uma palavra, mas de um caractere específico, ou pelo menos se a String contém esse determinado caractere. Vejamos como funciona isso através do seguinte código:

```
1  import java.util.Scanner;
2
3  public class ProgramaStringPosicaoChar {
4      public static void main(String[] args) {
5          Scanner leitor = new Scanner(System.in);
6          System.out.println("Digite uma palavra:");
7          String nome1 = leitor.next();
8          char ponto = '.';
9          System.out.println("A posição do ponto no texto é: "
10         + nome1.indexOf(ponto));
11     }
12 }
```

O programa mostrado faz uso da função **indexOf()**, a qual recebe um caractere como parâmetro e retorna a posição da primeira ocorrência desse caractere no texto representado no exemplo pela variável nome1. Caso o caractere não exista, a função irá retornar o valor -1. Também podemos verificar se existe mais de uma ocorrência. Veja agora como isso pode ser feito através do programa a seguir:

```java
import java.util.Scanner;

public class ProgramaStringPosicaoChar {
    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);
        System.out.println("Digite uma palavra:");
        String nome1 = leitor.next();
        char ponto = '.';
        int inicio = 0;
        int posicao = nome1.indexOf(ponto);
        System.out.println("Existe ponto nas posições: ");
        while (inicio < nome1.length() && posicao != -1) {
            System.out.println(posicao);
            inicio = posicao + 1;
            posicao = nome1.indexOf(ponto, inicio);
        }
    }
}
```

O ProgramaStringPosicaoChar faz uso de duas variáveis, uma chamada “inicio” e outra “posição”. A primeira variável é usada para indicar a partir de qual posição devemos buscar o caractere na String, por isso ela é inicializada com zero, o primeiro índice válido. Já a variável “posição” indica a localização do último caractere ponto encontrado. Ela é inicializada com o retorno da função **indexOf()**, ou seja, com o índice da primeira ocorrência do caractere procurado. Em seguida, temos um laço onde sua condição verifica se o início é maior ou igual ao tamanho da String (já percorremos toda a String?) ou se o valor da variável “posição” é -1 (o índice encontrado não é válido, então não há mais caractere procurado na String?).

Caso a execução do programa entre no laço, isso quer dizer que a variável “posição” tem um índice válido do caractere procurado. Para encontrar a posição do próximo caractere, vamos procurar a partir do índice da última ocorrência encontrada — mais um (posicao + 1). Por fim, realizamos a busca usando novamente a função indexOf(), só que agora passando como parâmetro não só o caractere procurado, mas também a variável “inicio”, indicando a partir de que posição na String (da esquerda para a direita) a busca deve ser realizada. Um pouquinho mais elaborado, mas funciona!

Atividade 05

1. Implemente e execute o ProgramaStringPosicaoChar, conforme mostrado na aula. Relate o comportamento do mesmo quando o usuário digita um texto contendo vários pontos.

Para encerrar a parte de busca em Strings, deixe-me lhe explicar como buscar por palavras ou subtextos em Strings. Imagine que você quer saber se um determinado nome digitado pelo usuário começa com João. Como você faria isso? Uma forma seria comparar caractere por caractere para ver se os quatro primeiros são as letras 'J', 'o', 'ã' e 'o'. Entretanto, existe uma forma mais simples, como mostrado no código a seguir:

```
1  import java.util.Scanner;
2
3  public class ProgramaSubstring {
4      public static void main(String[] args) {
5          Scanner leitor = new Scanner(System.in);
6          System.out.println("Digite seu nome completo:");
7          String nome = leitor.next();
8          if (nome.length() >=4 &&
9              nome.substring(0, 4).equalsIgnoreCase("João")) {
10             System.out.println("Olá João!");
11         } else {
12             System.out.println("Epa, você não é João!");
13         }
14     }
15 }
```

Observe que o ProgramaSubstring faz uso da função substring(), a qual retorna um pedaço da String original (ou seja, retorna uma substring) a partir de dois números. O primeiro deles representa o índice de início do texto. O segundo representa a posição de fim do texto + 1. Dessa forma, passar os números 0 e 4 como parâmetro indica que nós queremos o texto formado pelos caracteres da posição 0 a 3. Note que para fazer isso, precisamos testar se a String original possui pelo menos quatro caracteres.

Vídeo 04 - Substring

Atividade 06

1. Implemente e execute o ProgramaSubstring, conforme mostrado na aula. Relate o comportamento do mesmo quando o usuário digita um texto começando com João e outros começando com outro nome.

2. Altere o ProgramaSubstring, removendo a verificação de que o nome possui pelo menos 4 caracteres. Rode o programa e relate seu comportamento quando o usuário digita textos com menos de 4 caracteres.

4. Leitura complementar

Existem várias outras funções de manipulação de Strings. Leia sobre elas através do link abaixo, em especial as funções startsWith() e endsWith(). Essas funções podem economizar bastante o seu esforço de programação. De fato, o conhecimento das bibliotecas de programação é um dos fatores que tornam um programador mais produtivo do que outros.

- <http://download.oracle.com/javase/6/docs/api/java/lang/String.html>

5. Resumo

Nesta aula, você aprendeu que existem diversas funções para manipulação de Strings. Entre elas, está uma que transforma a String em um array de caracteres. Outras, porém, facilitam a manipulação das Strings como se elas fossem arrays. Você viu também como comparar Strings considerando ou não a sensibilidade a letras maiúsculas e minúsculas, bem como funções para passar a String toda para maiúscula ou minúscula. Sem falar nas funções de busca, que são várias, certo?

Para a próxima aula, falaremos sobre a criação de funções e procedimentos auxiliares, assunto bastante importante para quem quer desenvolver programas bem estruturados. Até lá!

6. Autoavaliação

1. Explique como podemos converter uma String para um array de caracteres.
2. Explique como podemos trabalhar com Strings como se elas fossem arrays de caracteres.
3. Explique como podemos fazer buscas dentro de uma String.

7. Referências

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. **Fundamentos da programação de computadores**: algoritmos, Pascal, C/C++ e Java. São Paulo: Editora Pearson, 2008.

THE JAVA tutorials. Disponível em: <http://download.oracle.com/javase/tutorial/>. Acesso em: 6 dez. 2011.