Java Developer

Java Core

Sumário

Objetivos deste Módulo

Ao final deste módulo, objetiva-se que o aluno adquira:

Conhecimentos:

- String
- StringBuffer x StringBuilder

1. String

Uma "String" é uma sequência de caracteres. Mais especificamente, é um objeto que representa uma série de caracteres alfanuméricos (letras, números, símbolos, etc.). Strings são muito utilizadas em programação para manipular texto e dados baseados em caracteres. Aqui estão algumas características importantes das Strings em Java:

Imutabilidade: Em Java, as Strings são imutáveis, o que significa que uma vez que uma String é criada, seu conteúdo não pode ser alterado. Se você desejar modificar uma String, na verdade, uma nova String será criada com as modificações desejadas.

Criação de Strings: Existem várias maneiras de criar objetos String em Java. Você pode usar aspas duplas para criar literais de string, como "Olá, mundo!". Você também pode criar Strings usando construtores, como new String("Exemplo"). No entanto, a forma mais comum de criar Strings é simplesmente atribuí-las a uma variável, como String meuTexto = "Isso é uma String";.

Concatenação: Você pode concatenar Strings usando o operador +. Por exemplo:

```
String primeiraParte = "Olá, ";
String segundaParte = "mundo!";
String saudacao = primeiraParte + segundaParte; // Isso resulta em "Olá, mundo!"
```

Métodos String: A classe java.lang.String oferece muitos métodos úteis para manipular Strings. Alguns exemplos incluem **length**() para obter o comprimento da String, **charAt**(index) para acessar caracteres individuais, **substring**(beginIndex, endIndex) para extrair partes da String, **toLowerCase**() e **toUpperCase**() para converter o caso dos caracteres, entre outros.

Comparação de Strings: Em Java, você não deve usar o operador == para comparar o conteúdo de duas Strings, pois ele compara as referências de objeto, não os valores reais. Em vez disso, você deve usar o método equals() para verificar se duas Strings têm o mesmo conteúdo.

Exemplo de comparação correta de Strings:

```
String str1 = "hello";
String str2 = "hello";
if (str1.equals(str2)) {
    System.out.println("As Strings são iguais.");
}
```

Escape Sequences: Em Strings, você pode usar sequências de escape para representar caracteres especiais, como aspas duplas (\"), barras invertidas (\\), caracteres de nova linha (\n), entre outros. Isso é útil para incluir caracteres que normalmente teriam um significado especial em uma String.
Exemplo de sequências de escape:

```
String caminho = "C:\\diretorio\\arquivo.txt";
String citacao = "Ela disse: \"Olá, mundo!\"";
```

String e Arrays de Caracteres: As Strings em Java são implementadas como um array de caracteres (char[]). Você pode acessar caracteres individuais de uma String usando o método charAt(index) ou converter uma String em um array de caracteres usando o método toCharArray().
Exemplo de acesso a caracteres em uma String:

```
String texto = "Exemplo";
char primeiroCaracter = texto.charAt(0); // 'E'
char[] caracteres = texto.toCharArray(); // ['E', 'x', 'e', 'm', 'p', 'l', 'o']

System.out.println(primeiroCaracter);
System.out.println(caracteres);
System.out.println(caracteres[1]);
```

Manipulação de Strings: A classe java.lang.String oferece uma variedade de métodos úteis para manipular e transformar Strings, como **trim**() para remover espaços em branco no início e no final, **replace**(oldChar, newChar) para substituir caracteres, **split**(delimiter) para dividir uma String em partes com base em um delimitador, entre outros.

```
String texto = "  Espaços   ";
String semEspacos = texto.trim(); // Remove espaços em branco
String substituido = texto.replace("Espaços", "Texto"); // Substitui "Espaços" por "Texto"
String[] partes = texto.split(" "); // Divide a String em partes com base em espaços em branco
```

Exemplos:

length(): Retorna o comprimento da String, ou seja, o número de caracteres na String.

```
String texto = "Hello, world!";
int tamanho = texto.length(); // Retorna 13
```

charAt(int index): Retorna o caractere na posição especificada pela variável index.

```
String texto = "Java";
char primeiroCaracter = texto.charAt(0); // Retorna 'J'
```

substring(int beginIndex): Retorna uma nova String que é uma subcadeia da String original, começando a partir do índice beginIndex.

```
String texto = "Exemplo";
String subString = texto.substring(3); // Retorna "mplo"
```

substring(int beginIndex, int endIndex): Retorna uma nova String que é uma subcadeia da String original, começando do índice beginIndex até (mas não incluindo) o índice endIndex.

```
String texto = "Exemplo";
String subString = texto.substring(1, 4); // Retorna "xem"
```

toLowerCase() e **toUpperCase**(): Converte a String em minúsculas ou maiúsculas, respectivamente.

```
String texto = "Texto";
String minusculas = texto.toLowerCase(); // Retorna "texto"
String maiusculas = texto.toUpperCase(); // Retorna "TEXTO"
```

equals(Object obj): Compara duas Strings para verificar se têm o mesmo conteúdo. Retorna true se forem iguais, caso contrário, retorna false.

```
String str1 = "hello";
String str2 = "Hello";
boolean saoIguais = str1.equals(str2); // Retorna false
```

equalsIgnoreCase(String outraString): Compara duas Strings, ignorando as diferenças entre maiúsculas e minúsculas.

```
String str1 = "hello";
String str2 = "Hello";
boolean saoIguais = str1.equalsIgnoreCase(str2); // Retorna true
```

contains(): Verifica se a String contém uma sequência especificada. Retorna true se a sequência estiver contida na String, caso contrário, retorna false.

```
String texto = "Olá, mundo!";
boolean contemMundo = texto.contains("mundo"); // Retorna true
```

replace(old, new): Substitui todas as ocorrências da sequência target pela sequência replacement na String.

```
String texto = "Hello, world!";
String novoTexto = texto.replace("Hello", "Hi"); // Retorna "Hi, world!"
```

trim(): Remove os espaços em branco no início e no final da String.

```
String texto = "   Espaços   ";
String semEspacos = texto.trim(); // Retorna "Espaços"
```

startsWith(String prefix) e **endsWith**(String suffix): Verifica se a String começa ou termina com a sequência especificada, respectivamente.

```
String texto = "Hello, world!";
boolean comecaComHello = texto.startsWith("Hello"); // Retorna true
boolean terminaComMundo = texto.endsWith("mundo"); // Retorna false
```

isEmpty(): Verifica se a String está vazia (não contém caracteres).

```
String vazia = "";
boolean estaVazia = vazia.isEmpty(); // Retorna true
```

indexOf(String str) e **lastIndexOf**(String str): Retorna a posição da primeira (ou última) ocorrência da sequência especificada na String, ou -1 se não for encontrada.

```
String texto = "Hello, world!";
int primeiraOcorrencia = texto.indexOf("o"); // Retorna 4
int ultimaOcorrencia = texto.lastIndexOf("o"); // Retorna 8
```

concat(String str): Concatena uma String à atual e retorna uma nova String.

```
String saudacao = "Olá, ";
String nome = "mundo!";
String mensagem = saudacao.concat(nome); // Retorna "Olá, mundo!"
```

2. StringBuffer x StringBuilder

StringBuffer e StringBuilder são duas classes em Java que servem para criar e manipular sequências **mutáveis** de caracteres (Strings). A principal diferença entre elas está na concorrência:

StringBuilder:

- Não é sincronizado: não é thread-safe, o que significa que não é seguro usá-lo em ambientes com várias threads (threads concorrentes).
- Mais eficiente: Por não ter as sobrecargas de sincronização, StringBuilder tende a ser mais rápido do que StringBuffer em operações simples de manipulação de Strings.
- Use quando a concorrência não é uma preocupação e você precisa de alta performance.

Exemplo de uso do StringBuilder:

```
StringBuilder sb = new StringBuilder();
sb.append("Olá, ");
sb.append("mundo!");
String resultado = sb.toString(); // Converte para uma String quando necessário
```

StringBuffer:

- Sincronizado: é thread-safe, o que significa que pode ser usado com segurança em ambientes com várias threads, pois as operações em StringBuffer são sincronizadas automaticamente para evitar conflitos de concorrência.
- Menos eficiente em cenários não concorrentes: Devido à sobrecarga de sincronização, StringBuffer pode ser um pouco mais lento do que StringBuilder em cenários onde a concorrência não é um problema.
- Use quando a concorrência é uma preocupação e você precisa de garantias de que as operações serão seguras em ambientes com várias threads.

Exemplo de uso do StringBuffer:

```
StringBuffer sb = new StringBuffer();
sb.append("Olá, ");
sb.append("mundo!");
String resultado = sb.toString(); // Converte para uma String quando necessário
```

3. Exercícios

1) Contar Ocorrências de uma Substring: escreva um programa Java que conte o número de ocorrências de uma substring em uma frase inserida pelo usuário. Faça teste unitário.

2) Inverter uma String: escreva um programa Java que inverta uma String inserida pelo usuário. Faça o teste unitário.

3) Separar Nomes em Sobrenomes: escreva um programa Java que receba um nome completo e separe o primeiro nome dos sobrenomes.

4) Conversão de Maiúsculas e Minúsculas: escreva um programa Java que permita ao usuário converter uma frase para letras maiúsculas ou minúsculas.

   Menu a ser criado:

   ```
   System.out.println("Escolha a opção:");
   System.out.println("1. Converter para maiúsculas");
   System.out.println("2. Converter para minúsculas");
   ```

5) Formatação de Nome: escreva um programa Java que receba o nome completo de uma pessoa (com primeiro nome, meio e sobrenome) e o formate como "Sobrenome, Primeiro Nome".

6) Substituir Palavras: Escreva um programa Java que substitua todas as ocorrências de uma palavra por outra em uma frase inserida pelo usuário.

   Menu a ser criado:

   ```
   System.out.print("Digite uma frase: ");
   String frase = scanner.nextLine();

   System.out.print("Digite a palavra a ser substituída: ");
   String palavraASubstituir = scanner.nextLine();

   System.out.print("Digite a palavra de substituição: ");
   String palavraSubstituta = scanner.nextLine();
   ```