04. Usando java.util.Scanner

Date	@17/10/2022
Categoria	Java
Curso	Java e java.io: Streams Reader e Writers

Revisão

Primeira data @February 21, 2022

Tópicos

- Leitura com Scanner
- Lendo arquivos com Scanner
- Parseando arquivo
- Separando o conteúdo
- Formatação de valores
- Formatação expressiva
- Mãos na massa: Scanner
- Para saber mais: java.util.Properties
- O que aprendemos?

Parseando arquivo

Ainda temos uma peculiaridade. No saldo, temos um ponto (`.`) separando os números inteiros dos decimais, contudo, alguns lugares convencionam o ponto, enquanto outros utilizam a vírgula (`,`) para este fim. O que determina se a máquina virtual seguirá um padrão ou outro é o sistema operacional da máquina, ela seguirá o padrão do idioma da máquina em que o código está sendo escrito.

Por exemplo, como neste curso estamos utilizando uma máquina cujo sistema está em Inglês, o padrão é o ponto (`.`), se tentarmos utilizar outro caractere, ocorrerá um erro.

Para evitar esta regra automática, podemos especificar no código a regra que queremos seguir utilizando o método `useLocale()`:

Nela, acessaremos o nome da classe, portanto `Locale`, seguida de um ponto (`.`), e o Eclipse nos mostrará uma série de opções de regras. No caso, utilizaremos `US`, que é o padrão americano:

```
//Código omitido

public class TesteLeitura2 {

	public static void main(String[] args) throws Exception {

		Scanner scanner = new Scanner(new File("contas.csv"));
		while(scanner.hasNextLine()) {
			String linha = scanner.nextLine();
			System.out.println(linha);

			Scanner linhaScanner = new Scanner(linha);
			linhaScanner.useLocale(Locale.US);
			linhaScanner.useDelimiter(",");

			String valor1 = linhaScanner.next();
			int valor2 = linhaScanner.nextInt();
			int valor3 = linhaScanner.nextInt();
			String valor4 = linhaScanner.next();
			double valor5 = linhaScanner.nextDouble();

			System.out.println(valor1 + valor2 + valor3 + valor4 + valor5);

			linhaScanner.close();

//			  String[] valores = linha.split(",");
//			  System.out.println(valores[1]);
		}
```

```
                scanner.close();
        }
}
```

Desta forma, não importa o sistema operacional, o código e a máquina virtual sempre respeitarão o padrão americano.

Salvaremos e executaremos, e o código continua funcionando perfeitamente.

Adiante, falaremos sobre a formatação.

Separando o conteúdo

Qual o método de Scanner responsável por separar o conteúdo por um delimitador?

Resposta: `scanner.useDelimiter(",");`

Para saber mais: java.util.Properties

Em projetos mais complexos (que você verá ainda em outros cursos) temos muitas configurações a fazer para nossa aplicação funcionar. Por exemplo, é preciso configurar usuários, senhas, endereços ou portas para acessar outros aplicativos e serviços. Um exemplo clássico é o acesso ao banco de dados, que precisa do login/senha, etc.

Essas configurações podem ficar dentro código fonte, mas isso exige a recompilação do código fonte assim que uma configuração muda. O melhor seria externalizá-las e colocá-las em um arquivo separado, por exemplo um arquivo de texto, que não exige compilação. Dessa forma só precisamos alterar esse texto, sem mexer no código fonte Java.

Ótimo, e exatamente isso é feito em milhares de projetos Java e para facilitar e padronizar mais ainda, foi criado um mini-padrão para esse tipo de arquivo. Eles são chamados de arquivos de propriedade ou simplesmente ***properties***.

Um arquivo *properties* associa o nome da configuração com o seu valor. Veja o exemplo:

```
login = alura
senha = alurapass
endereco = www.alura.com.br
```

A configuração `login` tem o seu valor `alura`, `senha` tem o valor `alurapass` e assim em diante. Sempre tem uma **chave** (*key*) e um **valor** (*value*) associados. E como isso é tão comum, já foi criado a classe específica `java.util.Properties`, para trabalhar com esses pares de chave/valor, mas claro, poderíamos usar um *Scanner* também!

O uso dessa classe é muito simples. Veja a representação dos valores acima, através de um objeto da classe `Properties`:

```
//import deve ser java.util.Properties
Properties props = new Properties();
props.setProperty("login", "alura"); //chave, valor
props.setProperty("senha", "alurapass");
props.setProperty("endereco", "www.alura.com.br");
```

Com o objeto criado podemos ver como escrever esses dados no HD.

Escrita de properties

Agora só falta gravar um arquivo para realmente externalizar as configurações. Para tal, você usa o método `store`, da classe `Properties`, que recebe um *stream* ou *writer*, além dos comentários desejados:

```
props.store(new FileWriter("conf.properties"), "algum comentário");
```

Isso cria um arquivo **conf.properties**, com os dados do objeto acima:

```
#algum comentário
#Thu May 10 14:29:38 BRT 2018
senha=alurapass
login=alura
endereco=www.alura.com.br
```

Leitura de properties

Para ler esse arquivo de *properties*, basta usar o método `load`:

```
Properties props = new Properties();
props.load(new FileReader("conf.properties"));

String login = props.getProperty("login");
String senha = props.getProperty("senha");
String endereco = props.getProperty("endereco");

System.out.println(login + ", " + senha  + ", " +  endereco);
```

Repare que, uma vez lido o arquivo, podemos usar o método `getProperty(key)`, da classe `Properties`, para recuperar o seu valor.

O que aprendemos?

Nesta aula aprendemos mais sobre a Scanner e também formatacão de texto. Segue a lista:

1. Leitura de arquivos com Scanner
2. Uso de delimitador com Scanner
3. Formatação de texto e também de números
4. Definição de Localização para formatar o texto.
