Linguagem de Programação I

Aula 9
Manipulação de Arquivos e Diretórios

l.bertholdo@ifsp.edu.br

Conteúdo

- Manipulação de Arquivos
- Manipulação de Diretórios
- Escrita em Arquivos
- Leitura de Arquivos
- Informações sobre Arquivos
- Informações sobre Diretórios

l.bertholdo@ifsp.edu.br

Manipulação de Arquivos e Diretórios

- Os programas apresentados até aqui armazenavam os dados de forma temporária, na **memória RAM** do computador. Com isso, ao fechar o programa, os dados eram perdidos.
- Contudo, em diversas situações é preciso “persistir” os dados da aplicação, ou seja, seus dados precisam ser mantidos na memória, mesmo após fechar a aplicação ou desligar o computador.
- Nesse caso, os dados são armazenados em uma **memória não volátil**, como: discos rígidos, cartões SSD (*solid-state drive*), *pendrives*, CDs, DVDs, etc.
- As formas mais comuns de armazenamento de dados não voláteis são os arquivos de sistema e os bancos de dados.

l.bertholdo@ifsp.edu.br

Manipulação de Arquivos e Diretórios

- Em uma aplicação, arquivos de texto são úteis para realização de tarefas simples que necessitam armazenar dados de forma não volátil, porém sem a necessidade de existir uma estrutura de banco de dados.

- Exemplos: armazenamento de *logs* de uma aplicação, troca de informações entre sistemas, armazenamento de textos em arquivos de localização (para alterar o idioma da aplicação) etc.

- As classes e métodos para acesso e manipulação de arquivos se encontram definidos no pacote **java.io** e **java.nio*** da API Java.

* O pacote **java.nio**, juntamente com os pacotes **java.nio.channels** e **java.nio.charset**, foram adicionados no **JDK 1.4** como um complemento do **java.io** original. E, a partir do **JDK 7**, foi incluído também o pacote **java.nio.file**.

l.bertholdo@ifsp.edu.br

Manipulação de Arquivos

- A classe **File** permite criar, excluir, renomear e mover arquivos e diretórios.
- Criação de arquivo

Aqui, a 1ª barra invertida serve para indicar que a 2ª barra é apenas um separador de diretórios, e não parte do caractere de escape "\t" (Tab).

```
// Cria um objeto para referenciar o arquivo a ser criado.
File arq = new File("C:\\Testes\\teste.txt");
if (!arq.exists()) { // Verifica se o arquivo não existe.
    // Cria o arquivo recebido no construtor da classe File.
    try {
        if (arq.createNewFile()) // Retorna true ou false.
            System.out.println("Arquivo criado!");
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
} else
    System.out.println("O arquivo já existe!");
```

O método **createNewFile** exige o lançamento da exceção **IOException** para tratar erros de entrada e saída de dados, por exemplo, um caminho inválido.

Manipulação de Arquivos

- Exclusão de arquivo

```
// Cria um objeto para referenciar o arquivo a ser excluído.
File arq = new File("C:\\Testes\\teste.txt");
if (arq.exists()) { // Verifica se o arquivo existe.
    // Exclui o arquivo recebido no construtor da classe File.
    if (arq.delete()) // Retorna true ou false.
        System.out.println("Arquivo excluído!");
} else
    System.out.println("O arquivo não existe!");
```

l.bertholdo@ifsp.edu.br

Manipulação de Arquivos

- Renomeação de arquivo

```
// Cria um objeto para referenciar o arquivo a ser renomeado.
File arq = new File("C:\\Testes\\teste.txt"); // Nome original
File arq2 = new File("C:\\Testes\\teste2.txt"); // Novo nome
if (arq.exists()) { // Verifica se o arquivo existe.
    // Renomeia o arquivo para o nome recebido no método renameTo.
    if (arq.renameTo(arq2)) // Retorna true ou false.
        System.out.println("Arquivo renomeado!");
} else
    System.out.println("O arquivo não existe!");
```

l.bertholdo@ifsp.edu.br

Manipulação de Arquivos

- Movimentação de arquivo

```
// Cria um objeto para referenciar o arquivo a ser movido.
File arq = new File("C:\\Testes\\teste.txt"); // Local original
File arq2 = new File("C:\\Testes\\Subtestes\\teste.txt"); // Novo local
if (arq.exists()) { // Verifica se o arquivo existe.
    // Move o arquivo para o local recebido no método renameTo.
    if (arq.renameTo(arq2)) // Retorna true ou false.
        System.out.println("Arquivo movido!");
} else
    System.out.println("O arquivo não existe!");
```

Manipulação de Diretórios

- Criação de diretório

```
// CRIAÇÃO DE DIRETÓRIO - MKDIR
// Cria um objeto para referenciar o diretório a ser criado.
File dir = new File("C:\\Testes\\Subtestes");
if (!dir.exists()) { // Verifica se o diretório não existe.
    // Cria o diretório Subtestes (os diretórios anteriores precisam existir),
    // caso contrário, o diretório não é criado, nem é gerada uma exceção.
    if (dir.mkdir()) // Retorna true ou false.
        System.out.println("Diretório criado!");
} else
    System.out.println("O diretório já existe!");
```

```
// CRIAÇÃO DE DIRETÓRIO - MKDIRS
// Cria um objeto para referenciar o(s) diretório(s) a ser(em) criado(s).
File dir = new File("C:\\Testes\\Subtestes");
if (!dir.exists()) { // Verifica se o diretório não existe.
    // Cria o diretório Subtestes, e os diretórios anteriores caso não existam.
    if (dir.mkdirs()) // Retorna true ou false.
        System.out.println("Diretório(s) criado(s)!");
} else
    System.out.println("O diretório já existe!");
```

l.bertholdo@ifsp.edu.br

Manipulação de Diretórios

- Exclusão de diretório

```
// Cria um objeto para referenciar o diretório a ser excluído (Subtestes).
File dir = new File("C:\\Testes\\Subtestes");
if (dir.exists()) { // Verifica se o diretório existe.
    // Exclui o diretório recebido no construtor da classe File.
    if (dir.delete()) // Retorna true ou false.
        System.out.println("Diretório excluído!");
    else
        System.out.println("O diretório não está vazio!");
} else
    System.out.println("O caminho ou o diretório não existe!");
```

l.bertholdo@ifsp.edu.br

Manipulação de Diretórios

- Renomeação de diretório

```
// Cria um objeto para referenciar o diretório a ser renomeado (Testes).
File dir = new File("C:\\Testes"); // Nome original
File dir2 = new File("C:\\Testes2"); // Novo nome
if (dir.exists()) { // Verifica se o diretório existe.
    // Renomeia o diretório para o nome recebido no método renameTo.
    if (dir.renameTo(dir2)) // Retorna true ou false.
        System.out.println("Diretório renomeado!");
    else
        System.out.println("O caminho de destino não existe ou o diretório "
                           + "de origem não está vazio!");
} else
    System.out.println("O caminho ou o diretório de origem não existe!");
```

Manipulação de Diretórios

- Movimentação de diretório

```
// Cria um objeto para referenciar o diretório a ser movido (Testes).
File dir = new File("C:\\Testes"); // Local original
File dir2 = new File("C:\\Fase1\\Testes"); // Novo local
if (dir.exists()) { // Verifica se o diretório existe.
    // Move o diretório para o local recebido no método renameTo.
    if (dir.renameTo(dir2)) // Retorna true ou false.
        System.out.println("Diretório movido!");
    else
        System.out.println("O caminho de destino não existe ou o diretório "
                           + "de origem não está vazio!");
} else
    System.out.println("O caminho ou o diretório de origem não existe!");
```

Escrita em Arquivos

- A classe **FileWriter** permite criar e abrir um arquivo para operações de escrita.
- A classe **BufferedWriter** cria um *buffer* (área de armazenamento temporário) para realizar operações de escrita em arquivos. Principais métodos:
  - **write** – Escreve uma única linha no arquivo e sem o caractere terminador de linha (\n).
  - **append** – Acrescenta uma linha após a última linha do arquivo.
  - **close** – Fecha o arquivo.

l.bertholdo@ifsp.edu.br

Escrita em Arquivos

teste.txt

```
Esta é a 1ª linha do arquivo.
Esta é a 2ª linha do arquivo.
Esta é a 3ª linha do arquivo.
Esta é a 4ª linha do arquivo.
Esta é a 5ª linha do arquivo.
Esta é a última linha do arquivo.
```

```java
try {
    // Cria e abre o arquivo especificado para a operação de escrita.
    // Caso o arquivo já exista, ele é sobrescrito.
    FileWriter fw = new FileWriter("C:\\Testes\\teste.txt");
    // Cria um buffer para realizar as operações de escrita no arquivo.
    BufferedWriter bw = new BufferedWriter(fw);
    System.out.println("Arquivo aberto!");
    for (int i = 0; i < 5; i++) // Escreve no arquivo.
        bw.write("Esta é a " + (i+1) + "ª linha do arquivo.\n");
    bw.append("Esta é a última linha do arquivo."); // Adiciona uma linha ao arquivo.
    bw.close(); // Fecha o arquivo.
    System.out.println("Arquivo fechado!");
} catch (IOException e) {
    System.out.println(e.getMessage());
}
```

```
Arquivo aberto!
Arquivo fechado!
```

l.bertholdo@ifsp.edu.br

Leitura de Arquivos

- A classe **FileReader** permite abrir um arquivo para operações de leitura.
- A classe **BufferedReader** cria um *buffer* (área de armazenamento temporário) para realizar operações de leitura em arquivos. Principais métodos:
  - **readLine** – Lê cada linha do arquivo de forma sequencial.
  - **close** – Fecha o arquivo.

l.bertholdo@ifsp.edu.br

Leitura de Arquivos

teste.txt

```
Esta é a 1ª linha do arquivo.
Esta é a 2ª linha do arquivo.
Esta é a 3ª linha do arquivo.
Esta é a 4ª linha do arquivo.
Esta é a 5ª linha do arquivo.
Esta é a última linha do arquivo.
```

```java
try {
    // Abre o arquivo especificado para a operação de leitura.
    FileReader fr = new FileReader("C:\\Testes\\teste.txt");
    // Cria um buffer para realizar as operações de leitura no arquivo.
    BufferedReader br = new BufferedReader(fr);
    System.out.println("Arquivo aberto!");
    String linha;
    while ((linha = br.readLine()) != null) // Lê uma linha do arquivo.
        System.out.println(linha);
    br.close(); // Fecha o arquivo.
    System.out.println("Arquivo fechado!");
} catch (FileNotFoundException e) {
    System.out.println(e.getMessage());
} catch (IOException e) {
    System.out.println(e.getMessage());
}
```

```
Arquivo aberto!
Esta é a 1ª linha do arquivo.
Esta é a 2ª linha do arquivo.
Esta é a 3ª linha do arquivo.
Esta é a 4ª linha do arquivo.
Esta é a 5ª linha do arquivo.
Esta é a última linha do arquivo.
Arquivo fechado!
```

l.bertholdo@ifsp.edu.br

Informações sobre Arquivos

- A classe **File** também também possui métodos para recuperar informações sobre um arquivo ou um diretório.
- Obtendo informações de um arquivo:

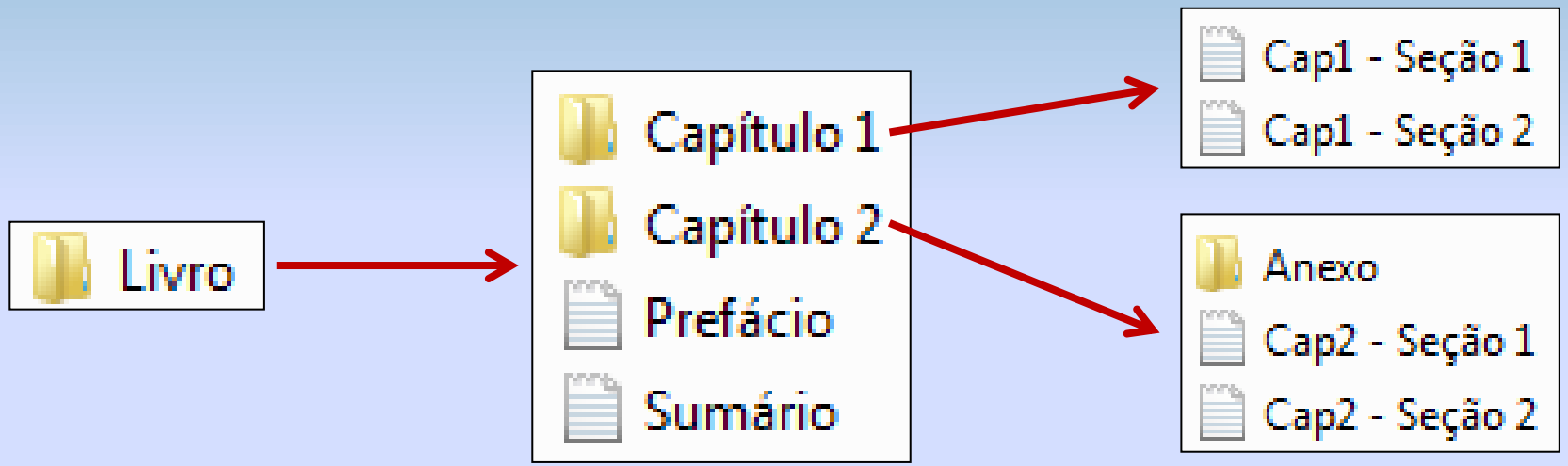
```
// Cria um objeto para referenciar o arquivo.
File arq = new File("C:\\Testes\\teste.txt");
if (arq.exists()) { // Verifica se o arquivo existe.
    Calendar cal = Calendar.getInstance(); // Cria um objeto Calendar.
    // Atribui ao objeto Calendar a data e a hora da última modificação no arquivo.
    cal.setTimeInMillis(arq.lastModified());
    // Obtém a data armazenada no objeto Calendar no formato "dd/MM/yyyy HH:mm:ss".
    String dataHora = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(cal.getTime());

    // Imprime as informações do arquivo.
    System.out.println("Última Modificação: " + dataHora);
    System.out.println("Caminho: " + arq.getPath());
    System.out.println("Diretório Pai: " + arq.getParent());
    System.out.println("Tamanho: " + arq.length() + " bytes");
} else
    System.out.println("O arquivo não existe!");
```

```
Última Modificação: 12/04/2017 22:45:23
Caminho: C:\Testes\teste.txt
Diretório Pai: C:\Testes
Tamanho: 183 bytes
```

Informações sobre Diretórios

- Obtendo informações de um diretório:



Informações sobre Diretórios

- Obtendo informações de um diretório:

```
Última Modificação: 12/04/2017 23:34:13
Caminho: C:\Livro
Diretório Pai: C:\
Arquivos e Diretórios Existentes:
Capítulo 1
Capítulo 2
Prefácio.txt
Sumário.txt
Arquivos e Diretórios Existentes
(com caminho):
C:\Livro\Capítulo 1
C:\Livro\Capítulo 2
C:\Livro\Prefácio.txt
C:\Livro\Sumário.txt
```

```
// Cria um objeto para referenciar o diretório.
File dir = new File("C:\\Livro");
if (dir.exists()) { // Verifica se o diretório existe.
    Calendar cal = Calendar.getInstance(); // Cria um objeto Calendar.
    // Atribui ao objeto Calendar a data e a hora da última modificação no diretório.
    cal.setTimeInMillis(dir.lastModified());
    // Obtém a data armazenada no objeto Calendar no formato "dd/MM/yyyy HH:mm:ss".
    String dataHora = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(cal.getTime());

    // Imprime as informações do diretório.
    System.out.println("Última Modificação: " + dataHora);
    System.out.println("Caminho: " + dir.getPath());
    System.out.println("Diretório Pai: " + dir.getParent());
    System.out.println("Arquivos e Diretórios Existentes: ");
    // Retorna apenas os arquivos e diretórios imediatamente abaixo do diretório informado.
    for (String a : dir.list())
        System.out.println(a);
    System.out.println("Arquivos e Diretórios Existentes\n(com caminho): ");
    // Retorna apenas os arquivos e diretórios imediatamente abaixo do diretório informado.
    for (File a : dir.listFiles())
        System.out.println(a);
} else
    System.out.println("O diretório não existe!");
```

Referências

- Peter Jandl Junior; Java – Guia do Programador – 3ª Edição. São Paulo: Novatec Editora, 2015.

- Sandra Puga, Gerson Risseti; Lógica de Programação e Estruturas de Dados com Aplicações em Java. Pearson Prentice Hall, 2006.