Manipulacao de Diretorios e Arquivos em Java

Do Básico ao Backup Completo - Java 8, 11 e 17

A manipulação de diretórios e arquivos é uma tarefa comum e necessária em muitos projetos.

Com Java, podemos criar diretórios, gerar documentos, mover arquivos entre pastas e realizar backups completos compactando pastas inteiras.

Este artigo é um guia prático para essas operações utilizando **Java 8**, **Java 11** e **Java 17**.

Aqui veremos como:

- *Criar pastas em Java*
- *Gerar arquivos Word e Excel*
- *Mover arquivos entre pastas*
- *Compactar e fazer backup de uma pasta inteira*

Todos os exemplos foram testados e são compatíveis com Java 8, 11, e 17.

Criando Pastas em Java

Criar um diretório em Java é simples com a classe Files da biblioteca java.nio.file.

O método utilizado é o Files.createDirectory(), que cria o diretório especificado caso ele ainda não exista.

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.io.IOException;

public class CriarPasta {
    public static void main(String[] args) {
        Path caminhoPasta = Paths.get("meuDiretorio");

        try {
            if (!Files.exists(caminhoPasta)) {
```

```
            Files.createDirectory(caminhoPasta);
            System.out.println("Pasta criada: " + caminhoPasta.toAbsolutePath());
        } else {
            System.out.println("Pasta já existe.");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
  }
}
```

- **Files.createDirectory()**: Cria a pasta especificada no caminho.
- **Files.exists()**: Verifica se a pasta já existe antes de criá-la, evitando exceções.

Gerando Arquivos Word com Apache POI

Para criar arquivos Word em Java, usamos a biblioteca **Apache POI**.

Ela permite a criação e manipulação de documentos do Microsoft Office, como arquivos .docx.

Adicione a dependência do Apache POI ao seu projeto (exemplo para Maven):

```
<dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi-ooxml</artifactId>
  <version>5.2.3</version>
</dependency>
```

Agora, o código para criar um arquivo Word:

```
import org.apache.poi.xwpf.usermodel.XWPFDocument;
import java.io.FileOutputStream;
import java.io.IOException;

public class CriarWord {
  public static void main(String[] args) {
    try (XWPFDocument documento = new XWPFDocument()) {
      FileOutputStream out = new FileOutputStream("meuDocumento.docx");
      documento.createParagraph().createRun().setText("Conteúdo do meu arquivo Word");
      documento.write(out);
      out.close();
      System.out.println("Arquivo Word criado com sucesso!");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

- **XWPFDocument**: Representa o documento Word.
- **createParagraph().createRun().setText()**: Adiciona texto ao documento.
- **FileOutputStream**: Utilizado para gravar o arquivo .docx.

Gerando Arquivos Excel com Apache POI

Também podemos usar a biblioteca **Apache POI** para gerar arquivos Excel.

O código abaixo cria uma planilha com uma célula preenchida.

```
import org.apache.poi.ss.usermodel.*;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import java.io.FileOutputStream;
import java.io.IOException;

public class CriarExcel {
    public static void main(String[] args) {
        Workbook workbook = new XSSFWorkbook();
        Sheet sheet = workbook.createSheet("Planilha1");

        Row row = sheet.createRow(0);
        Cell cell = row.createCell(0);
        cell.setCellValue("Conteúdo da célula");

        try (FileOutputStream out = new FileOutputStream("meuArquivo.xlsx")) {
            workbook.write(out);
            workbook.close();
            System.out.println("Arquivo Excel criado com sucesso!");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- **Workbook**: Representa a pasta de trabalho Excel.
- **createSheet()**: Cria uma nova planilha.
- **createRow() e createCell()**: Cria uma nova linha e célula, respectivamente.
- **setCellValue()**: Define o valor da célula.

Movendo Arquivos Entre Pastas

Para mover arquivos entre pastas, utilizamos o método Files.move() da API java.nio.file.

Ele também permite substituir um arquivo existente no destino, se necessário.

```
import java.nio.file.*;

public class MoverArquivo {
    public static void main(String[] args) {
        Path origem = Paths.get("meuDiretorio/meuArquivo.txt");
        Path destino = Paths.get("novoDiretorio/meuArquivo.txt");

        try {
            Files.move(origem, destino, StandardCopyOption.REPLACE_EXISTING);
            System.out.println("Arquivo movido com sucesso!");
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- **Files.move()**: Move o arquivo do diretório de origem para o diretório de destino.
- **StandardCopyOption.REPLACE_EXISTING**: Sobrescreve o arquivo de destino, caso ele já exista.

Compactando uma Pasta Inteira em um Arquivo ZIP (Backup)

Uma maneira eficiente de realizar backups é compactar todos os arquivos de um diretório em um arquivo ZIP. A classe ZipOutputStream de java.util.zip facilita esse processo.

```
import java.io.*;
import java.nio.file.*;
import java.util.zip.*;

public class CompactarDiretorio {
    public static void main(String[] args) {
        String origem = "meuDiretorio";
        String destinoZip = "backup.zip";

        try (FileOutputStream fos = new FileOutputStream(destinoZip);
             ZipOutputStream zipOut = new ZipOutputStream(fos)) {

            Path origemPath = Paths.get(origem);
            Files.walk(origemPath).filter(path -> !Files.isDirectory(path)).forEach(path -> {
                try (FileInputStream fis = new FileInputStream(path.toFile())) {
                    String zipEntryName = origemPath.relativize(path).toString();
                    zipOut.putNextEntry(new ZipEntry(zipEntryName));
                    byte[] bytes = new byte[1024];
                    int length;
                    while ((length = fis.read(bytes)) >= 0) {
                        zipOut.write(bytes, 0, length);
                    }
                    zipOut.closeEntry();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            });

            System.out.println("Backup criado com sucesso: " + destinoZip);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- **Files.walk()**: Percorre todos os arquivos e diretórios de forma recursiva.
- **ZipOutputStream**: Escreve os arquivos no formato ZIP.
- **ZipEntry**: Representa cada arquivo compactado.

Conclusão

Neste artigo, mostramos como manipular diretórios e arquivos em **Java 8**, **Java 11** e **Java 17**, abordando desde a criação de pastas e arquivos Word/Excel até o backup de uma pasta inteira compactada em ZIP.

As APIs do Java tornam essas tarefas eficientes e, com a ajuda de bibliotecas como o **Apache POI**, podemos manipular facilmente documentos do Microsoft Office.

Ao longo das três versões do Java, a abordagem para manipulação de arquivos e diretórios permanece consistente, com algumas melhorias de performance e segurança nas versões mais recentes.

Esses exemplos podem ser aplicados em qualquer projeto que envolva manipulação de arquivos e pastas, garantindo a integridade e organização dos dados.

EducaCiência FastCode para a comunidade