# Manipulação de Arquivos

Sugiro consultar <https://docs.oracle.com/javase/tutorial/essential/io/fileio.html> para serviços oferecidos pela biblioteca Java padrão, além de um uso comentado, em particular, para grandes arquivos ([aqui](https://bit.ly/2I45GHv)).

Ler o conteúdo de um arquivo e escrever em um arquivo são operações comuns. Abaixo são fornecidos vários cenários nos quais arquivos são processados.

## Criar arquivo texto

A criação de um simples arquivo texto pode ser realizada conforme abaixo.

**public static void** main(String[] args) **throws** FileNotFoundException,  
 UnsupportedEncodingException {

PrintWriter pw = **new** PrintWriter(args[0], **"UTF-8"**);

pw.write(args[1]);

pw.close();

}

Nesse caso, o primeiro argumento fornecido ao programa será utilizado como o nome do arquivo no qual será depositado o conteúdo fornecido como o segundo argumento.

## Exibir o conteúdo de arquivo texto

Enquanto o trecho de código acima cria um arquivo texto, usando o padrão UTF-8 para codificar o texto, aquele abaixo realiza a operação inversa, carrega o conteúdo do arquivo e exibe, linha por linha, o conteúdo do mesmo.

Charset uft8 = Charset.*forName*(**"UTF-8"**);

Path arquivo = Paths.*get*(args[0]);

**for** (String linha : Files.*readAllLines*(arquivo, uft8)) {

System.***out***.println(linha);

}

## Exibir o conteúdo de arquivo texto (alternativa)

Em alguns casos simples pode-se carregar o conteúdo do arquivo e exibi-lo, linha por linha, conforme ilustrado no cenário anterior. Em outros, é preciso carregar cada linha e exibi-la, linha após linha, o que permite exibir o conteúdo de arquivos texto de tamanho arbitrário. Essa alternativa é ilustrada abaixo.

FileInputStream fis = **new** FileInputStream(args[0]);

InputStreamReader isr = **new** InputStreamReader(fis, **"UTF-8"**);

BufferedReader br = **new** BufferedReader(isr);

String linha;

**while** ((linha = br.readLine()) != **null**) {

System.***out***.println(linha);

}  
  
br.close();

Um trecho de código similar àquele acima é fornecido abaixo.

Path arquivo = Paths.*get*(args[0]);

Charset utf8 = Charset.*forName*(**"UTF-8"**);

BufferedReader br = Files.*newBufferedReader*(arquivo, utf8);

String linha;

**while** ((linha = br.readLine()) != **null**) {

System.***out***.println(linha);

}

## Criar arquivo binário

O conteúdo de um arquivo não necessariamente é formado apenas por texto. Um arquivo contendo apenas valores inteiros pode ser criado usando os bytes correspondentes a cada inteiro em vez da representação textual. Por exemplo, em vez do texto “2” seriam utilizados quatro bytes, três deles com o valor zero e o quarto com o valor hexadecimal 2. O trecho abaixo ilustra como criar um arquivo contendo um inteiro.

FileOutputStream fos = new FileOutputStream(args[0]);

DataOutputStream dos = new DataOutputStream(fos);

dos.writeInt(inteiro);

dos.close();

## Exibe conteúdo de arquivo binário

O trecho de código abaixo recupera do arquivo indicado um inteiro, contido nos quatro primeiros bytes. Ele realiza a operação inversa ao código acima, ou seja, o valor exibido deve ser o mesmo daquele recuperado pelo código abaixo.

FileInputStream fis = new FileInputStream(args[0]);

DataInputStream dis = new DataInputStream(fis);

int valor = dis.readInt();

dis.close();

System.out.println(valor);

Na linha acima, em vez de simplesmente exibir o valor inteiro em decimal, você pode usar o método abaixo para realizar a exibição em hexadecimal.

System.out.println(Integer.toHexString(valor));

## Criação via buffer

Também é possível fazer uso de um buffer no qual o valor a ser armazenado é depositado e só então enviado para o *Stream* de saída, conforme ilustrado abaixo.

Path arquivo = Paths.*get*(args[0]);

int valor = Integer.*parseInt*(args[1]);

ByteBuffer buffer = ByteBuffer.*allocate*(10);

buffer.putInt(valor);

buffer.flip();

FileChannel fc = FileChannel.*open*(arquivo, *CREATE*, *WRITE*);

fc.write(buffer);

fc.close();

## Carregamento via buffer

O processo de leitura do conteúdo de um *Stream* também pode fazer uso de um *buffer*, conforme ilustrado abaixo.

Path arquivo = Paths.*get*(args[0]);

ByteBuffer buffer = ByteBuffer.*allocate*(10);

FileChannel fc = FileChannel.*open*(arquivo, ***READ***);

fc.read(buffer);

fc.close();

buffer.flip();

**int** valor = buffer.getInt();

System.***out***.println(valor);

## Percorrendo diretórios

O trecho de código abaixo exibe todos os diretórios e subdiretórios que podem ser alcançados a partir do diretório corrente.

public class ListarDiretorios extends   
 SimpleFileVisitor<Path> {

@Override

public FileVisitResult postVisitDirectory(  
 Path dir, IOException exc) throws IOException {

System.out.println(dir.toAbsolutePath());

return FileVisitResult.CONTINUE;

}

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

Path path = Paths.get(".");

ListarDiretorios dld = new ListarDiretorios();

Files.walkFileTree(path, dld);

}

}