[**Aprendendo a manipular arquivos**](http://www.guj.com.br/articles/13)

em 05/09/2002 , por [Guilherme Silveira](mailto:guilherme@guj.com.br)

**Introdução**

Uma das principais facilidades em Java, comparando com outras linguages de programação, é a facilidade na leitura e gravação de arquivos no sistema operacional sem ter de se preocupar com o sistema operacional no qual sua aplicação está rodando. Sendo Java uma linguagem orientada a objetos, nada mais claro que utilizar classes e instâncias delas (objetos) para lidar com a saida e entrada de dados (input/output, abreviado: io). O java trata a entrada e saída como fluxos de dados (os tão chamados Streams), que você tem pleno controle sobre eles. Além disso, a abstração criada pela linguagem sobre os Streams é tão grande, que muitas vezes você está puxando/escrevendo dados em algum Stream, e você não sabe se eles estão vindo da internet, de um arquivo texto, ou do usuário que está digitando no console. E o mais importante: não faz a menor diferença esta infromação! As classes ligadas a *io* estão nos pacotes *java.io* e *java.nio* (NEW io) e estarei ensinando o básico das classes responsáveis pela leitura e gravação de arquivos de modo não binário, isto é, tratando com Strings e caracteres. Este tutorial explicará passo a passo o que acontece no código, para que você entenda perfeitamente o que é feito, e não decore simplesmente o código.

**java.io.File**

Instâncias da classe *java.io.File* representam caminhos (paths) para possíveis locais no sistema operacional. Lembre-se que ele apenas representa um arquivo ou diretório, isto não quer dizer que este caminho exista ou não. Por exemplo, o código a seguir cria uma instância da classe mencionada, que aponta para */usr/guj/arquivo.txt*:

[view plainprint?](http://www.guj.com.br/articles/13)

1. File file = new File("/usr/guj/arquivo.txt");

E o código a seguir um caminho para o arquivo *arquivo.txt* que estara no diretório atual do sistema e *../arquivo.txt* que estará no diretório pai do atual. **Dica:** No caso do sistema operacional windows, você deve usar duplas barras invertidas (\\), já que uma barra invertida apenas é considerado escape pelo java! **Dica:** Em java, voce NÃO tem como alterar o diretório atual, por isso pode criar instâncias da classe *File* que apontam para lugares relativos, mas sempre ao diretório inicial, veja o segundo exemplo abaixo:

[view plainprint?](http://www.guj.com.br/articles/13)

1. File noDiretorioAtual = new File("arquivo.txt");
2. File noDiretorioAnterior = new File("../arquivo.txt");

Existe outras maneiras de criar caminhos relativos a um outro caminho, por exemplo:

[view plainprint?](http://www.guj.com.br/articles/13)

1. File diretorioRaiz = new File("/");
2. File arquivo1 = new File(diretorioRaiz,"autoexec.bat");
3. File arquivo2 = new File(diretorioRaiz,"config.sys");
4. File diretorioWindows = new File(diretorioRaiz,"windows");
5. File diretorioWindows2 = new File("/windows/");
6. File diretorioWindows3 = new File("/windows");
7. File diretorioWindows4 = new File("c:\\\\windows");

**Dica:** É importante saber que criar um ponteiro para um caminho no sistema operacional não significa criar o arquivo/diretório. As instanciações do código acima iriam funcionar mesmo se você não possuisse esses arquivos, porém não irão apontar para um arquivo/diretório existente.

**Um pouco mais da API da classe java.io.File**

Vamos mostrar um pouco mais da API desta classe, para que você saiba o que tem disponível. O primeiro passo é descobrir se uma instância da classe *java.io.File* aponta para algo que existe, para um diretório ou arquivo. Os três métodos úteis nesse momento são:

[view plainprint?](http://www.guj.com.br/articles/13)

1. boolean exists();
2. boolean isDirectory();
3. boolean isFile();

Outros métodos que podem ser úteis para checar a viabilidade de leitura e gravaçã sao:

[view plainprint?](http://www.guj.com.br/articles/13)

1. boolean canRead();
2. boolean canWrite();

E por fim, alguns métodos de criação de diretórios, sendo o primeiro o que cria um diretório e o segundo cria todos os diretórios necessarios para chegar no caminho apontado pelo objeto. Esses costumam ser muito utilizados:

[view plainprint?](http://www.guj.com.br/articles/13)

1. boolean mkdir();
2. boolean mkdirs();

E outros métodos comuns são:

[view plainprint?](http://www.guj.com.br/articles/13)

1. boolean renameTo(File file);
2. long length();
3. long lastModified();
4. boolean delete();

Todos os nomes são auto explicativos. Qualquer coisa, consulte a API da classe *File*, já que esta será sua eterna companheira. Alguma pessoas criticam muito essa classe, pois ela deixa você instanciá-la mesmo que ela não represente um arquivo realmente, o que não acontece com as classes a seguir.

**java.io.FileWriter, java.io.PrintWriter**

Nesse tutorial não discutiremos sobre arquivos que contém todo o leque de caracteres Unicode e sim arquivos ASCII-8, portanto, aqueles que procuram esse tipo de informação, podem dar uma olhada nas classes *java.io.FileOutputStream* e *java.io.OutputStreamWriter* que irão criar um Writer para ser encapsulado pelo PrintWriter. Para escrever (e ler de) em um arquivo precisamos saber que Java tem duas camadas de io. Um baixo que le e escreve *bytes* e um alto que escreve e le caracteres. Vamos estudar os de alto nível, juntamente com algum encapsulando por cima deste, pois o de alto nível mexe com *char*s, e o que encapsula mexe com tudo! O rescritor de caracteres que iremos usar será uma instância da classe *java.io.FileWriter*:

[view plainprint?](http://www.guj.com.br/articles/13)

1. FileWriter writer = new FileWriter(new File("saida.txt"),true);
2. FileWriter writer = new FileWriter(new File("saida.txt"));
3. FileWriter writer = new FileWriter("saida.txt",true);
4. FileWriter writer = new FileWriter("saida.txt");

Repare que as duas primeiras linhas obtém o mesmo resultado que a terceira e a quarta, pois existem construtores para objetos do tipo *File* e *String*. **Dica:** Se você especificar um segundo parametro como *true* (primeira e terceira linha), você pode dizer que deseja adicionar dados a um arquivo que ja existe em vez de sobregravar esse arquivo (modo append). Agora já temos uma instância do "gravador" de caracteres, mas desejamos escrever Strings para o arquivo utilizando uma instancia da classe *java.io.PrintWriter* que fornece as funcoes *print()* e *println(0*.

[view plainprint?](http://www.guj.com.br/articles/13)

1. PrintWriter saida = new PrintWriter(writer);
2. PrintWriter saida = new PrintWriter(writer,true);

**Dica:** especificando o segundo parametro como true, os dados serão enviados para o arquivo a toda chamada do método *println()*, caso contrário, os dados só são enviados quando voce enviar uma quebra de linha, fechar o arquivo ou mandar ele atualizar as mudanças (modo autoflush). Repare que você utiliza um objeto da classe *java.io.PrintWriter* sempre! o *System.out* é uma instância desta classe, obviamente encapsulando uma outra Stream. Agora você já pode escrever no arquivo que abriu:

[view plainprint?](http://www.guj.com.br/articles/13)

1. saida.println("Camera's ready,");
2. saida.println("Prepare to flash....");
3. saida.println("Green Velvet");

Por fim, voce deve fechar o arquivo na ordem contrária que abriu, por segurança:

[view plainprint?](http://www.guj.com.br/articles/13)

1. saida.close();
2. writer.close();

**Dica:** Nunca se esqueça de fechar esses objetos, que liberam recursos para o sistema. Esperar pelo Garbage Collector pode ser a diferença da sua aplicação rodar rápido ou não!

**java.io.FileReader e java.io.BufferedReader**

*Assim como sobre a escrita, vocÊ deve dar uma lida na API das classes java.io.FileInputStream e java.io.InputStreamWriter para utilizar de métodos compatíveis com caracteres UNICODE.* O leito de caracteres que iremos usar durante a leiturá sera uma instância da classe *java.io.FileReader*:

[view plainprint?](http://www.guj.com.br/articles/13)

1. FileReader reader = new FileReader(new File("saida.txt"));
2. FileReader reader = new FileReader("saida.txt");

Como no exemplo de escrita, as duas linhas acima possuem o mesmo resultado. Agora, instanciamos um objeto da classe *java.io.BufferedReader* que fornece a funcao *readLine()* e algumas outras, para agilizar o trabalho.

[view plainprint?](http://www.guj.com.br/articles/13)

1. BufferedReader leitor = new BufferedReader(reader);
2. BufferedReader leitor = new BufferedReader(reader,1\*1024\*1024);

**Dica:** voce pode especificar o tamanho do buffer desejado no segundo parâmetro. Esse buffer é utilizado para minimizar o número de pedidos de io para ler blocos maiores do arquivo de uma vez só. Aumentando esse número diminui os acessos a disco, mas aumenta o consumo de memória! Agora você já pode ler:

[view plainprint?](http://www.guj.com.br/articles/13)

1. String linha = null;
2. while((linha = leitor.readLine())) {
3. System.out.println("Linha: " + linha);
4. }

E não se esqueça de fechar os dois objetos:

[view plainprint?](http://www.guj.com.br/articles/13)

1. leitor.close();
2. reader.close();

**java.io.BufferedWriter**

Você pode utilizar a classe *BufferedWriter* para agilizar também o processo de gravação:

[view plainprint?](http://www.guj.com.br/articles/13)

1. PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter("saida.txt")));
2. PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter("saida.txt"),1\*1024\*1024));

Caso voce não utilize um objeto BufferedWriter no meio, voce terá como resultado uma conversão de caracteres em bytes e gravação no arquivo a cada chamada do método *print* e suas variantes, o que pode ser MUITO custoso.

**Resumindo**

Agora você já consegue ler e gravar arquivos, inclusive otimizando as respectivas operações. Os próximos passo podem ser:

 aprender a gravar e ler objetos de um arquivo utilizando as classes *ObjectInputStream* e *ObjectOutputStream*,

 acessar um arquivo de modo aleatório para leitura e gravaçã usando a classe *RandomAccessFile*,

 parsear a Stream usando (a super lenta) *StreamTokenizer*,

 ler e gravar em variaveis em vez de arquivos usando *StringReader* e *StringWriter*,

 utilizar caracteres UNICODE especificando a codificacao ao instanciar objetos do tipo *InputStreamReader* e *InputStreamWriter*. De exercício, fica ler um arquivo e gravar uma cópia dele alterando todas as aparições de uma determinada String em uma outra, da maneira mais eficiente possível!