Manipulação de Arquivos

SCC0604 - Programação Orientada a Objetos

Prof. Fernando V. Paulovich
http://www.icmc.usp.br/~paulovic
paulovic@icmc.usp.br

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

18 de novembro de 2010

Sumário

Sumário

Introdução

- Java oferece várias classes para escrita e leitura de arquivos
- A seguir vamos ver algumas delas

Tipos de Arquivos

- Os arquivos podem ser classificados em arquivos de texto ou arquivos binários
  - Arquivos de texto: são compostos por uma série de caracteres *ASCII* agrupados em uma ou mais linhas. São compreendidos pelos seres humanos
  - Arquivos binários: composto por uma série de **bytes** representados por caracteres não compreendidos pelo ser humano. São menores que os arquivos de texto. Ex.: imagens, vídeo, áudio, etc

Manipulação de Arquivos

- Pacote **java.io** possui as classes para a manipulação de arquivos
- Essas classes são divididas em duas hierarquias de acordo com o tipo de arquivos que manipulam
  - **FileInputStream/FileOutputStream** (arquivos binários)
  - **FileReader/FileWriter** (arquivos de texto)
- Os arquivos e diretórios podem ser representados através da classe **File**

Classe File

```
1  public list(); //retorna lista de arquivos contidos no diretório
2  public boolean isFile(); //retorna se é um arquivo
3  public boolean isDirectory(); //retorna se é um diretório
4  public boolean delete(); //tenta apagar o diretório ou arquivo
5  public long length(); //retorna o tamanho do arquivo em bytes
6  public boolean mkdir(); //cria um diretório com o nome do arquivo
7  public String getAbsolutePath(); //retorna o caminho absoluto
8  public String getPath(); //retorna o caminho
9  public String getName(); //retorno o nome do arquivo
10 ...
```

Classe File

```
1  File dir = new File("dir");
2  boolean res = dir.mkdir(); //cria diretório
3
4  if (res || dir.exists()) { //verifica se criou ou se já existe
5    File subdir = new File(dir, "subdir");
6    subdir.mkdir(); //cria subdiretório
7
8    File arq = new File(dir, "arquivo.txt");
9    arq.createNewFile(); //cria um arquivo vazio
10
11   File[] arqs = dir.listFiles(); //retorna a lista de arquivos
12   for (File f : arqs) {
13     System.out.println(f);
14   }
15
16   //apagando o que foi criado
17   for (File f : arqs) {
18     f.delete();
19   }
20   dir.delete(); //diretório tem que estar vazio
21 }
```

Sumário

Classe FileReader

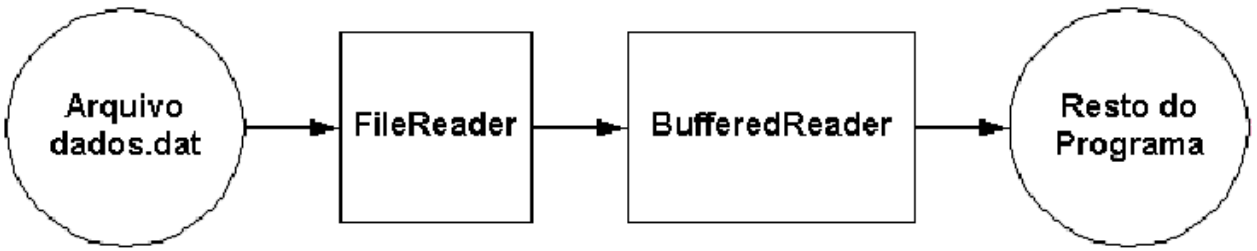
- Utilizada para escrita em arquivos de texto
- Construtores
  - public FileReader(String name)
  - public FileReader(File file)
- Usa o método **read()** para ler um caracter por vez

Classe FileReader

```
1  FileReader reader = new FileReader(new File("Main.java"));
2
3  int c;
4  while((c = reader.read()) != -1) {
5    System.out.print((char)c);
6  }
7
8  reader.close();
```

Classe BufferedReader

- Para acelerar a leitura é usada a classe **BufferedReader**



Classe BufferedReader

- Para acelerar a leitura é usada a classe **BufferedReader**

```
1  FileReader reader = new FileReader(new File("Main.java"));
2  BufferedReader breader = new BufferedReader(reader);
3
4  String linha = null;
5  while ((linha = breader.readLine()) != null) {
6    System.out.println(linha);
7  }
8
9  breader.close();
10 reader.close();
```

Classe StringTokenizer

- Para pegar palavras individuais de uma linha é possível usar a classe **StringTokenizer**
- O delimitador das palavras é informado no construtor da classe **StringTokenizer**

```
1  FileReader reader = new FileReader(new File("Main.java"));
2  BufferedReader breader = new BufferedReader(reader);
3
4  String linha = null;
5  while ((linha = breader.readLine()) != null) {
6    //O primeiro argumento é a string e o segundo é o delimitador
7    StringTokenizer st = new StringTokenizer(linha, " ");
8
9    while(st.hasMoreTokens()) {
10     System.out.print(st.nextToken());
11   }
12
13   System.out.println();
14 }
15
16 breader.close();
17 reader.close();
```

Classe FileWriter

- Utilizada para escrita em arquivos de texto
- Construtores
  - public FileWriter(String name)
  - public FileWriter(String name, boolean append)
  - public FileWriter(File file)
  - public FileWriter(File file, boolean append)
- Usa o método **write()** para escrever um caractere por vez ou uma *string* por vez

Classe FileWriter

```
1  FileWriter writer = new FileWriter(new File("teste.txt"));
2
3  char c;
4  while ((c = (char) System.in.read()) != '\n') {
5    writer.write(c);
6  }
7
8  writer.close(); //nunca esquecer de fechar o arquivo
```

Classe BufferedWriter

- Para agilizar a escrita é utilizada a classe **BufferedWriter**

```
1  FileWriter writer = new FileWriter(new File("teste.txt"));
2  BufferedWriter bwriter = new BufferedWriter(writer);
3
4  bwriter.write("escrever 1a linha\r\n");
5  bwriter.write("escrever 2a linha\r\n");
6
7  bwriter.flush(); //descarrego o buffer
8
9  bwriter.write("escrever 3a linha\r\n");
10 bwriter.write("escrever 4a linha\r\n");
11
12 bwriter.close(); //nunca esquecer de fechar o arquivo
13 writer.close(); //nunca esquecer de fechar o arquivo
```

Sumário

Classe FileOutputStream

- Utilizada para escrita em arquivos binários
- Construtores
  - public FileOutputStream(String name)
  - public FileOutputStream(String name, boolean append)
  - public FileOutputStream(File file)
  - public FileOutputStream(File file, boolean append)

Classe FileOutputStream

```
1 FileOutputStream fos = new FileOutputStream(new File("teste.bin"));
2
3 byte[] stream = new byte[]{'l','i','x','o'};
4 fos.write(stream); //escreve vetor de bytes
5
6 fos.close();
```

Classe FileInputStream

- Utilizada para leitura de arquivos binários
- Construtores
  - public FileInputStream(String name)
  - public FileInputStream(File file)

Classe FileInputStream

```
1  FileInputStream fis = new FileInputStream(new File("teste.bin"));
2
3  byte[] stream = new byte[100];
4  int tam = fis.read(stream); //retorna quantos bytes foram lidos
5
6  for(int i=0; i < tam; i++) {
7    System.out.print((char)stream[i]);
8  }
9
10 fis.close();
```

Classe RandomAccessFile

- Classe que permite a leitura e escrita em um arquivo com acesso randômico
- Possui um *file pointer* que indica a posição atual para acessar o arquivo
- O *file pointer* pode ser obtido através do método **getFilePointer()** e alterado através do método **seek()**

Classe RandomAccessFile

```
1  //Último parâmetro é o modo de abertura "r", "w", "rw", etc.
2  RandomAccessFile r = new RandomAccessFile(new File("teste.bin"), "r");
3
4  int posini = 2; //posição inicial de leitura
5  r.seek(posini); //posiciona o ponteiro de leitura
6
7  //Retorna o número de bytes no arquivo
8  for (int i = 0; i < r.length() - posini; i++) {
9    char c = (char) r.readByte();
10   System.out.print(c);
11 }
12
13 r.close();
```

Sumário

Leitura/Gravação Objetos

- Java permite a gravação direta de objetos em disco ou seu envio através da rede
  - Para isto, o objeto deve declarar implementar **java.io.Serializable**

Leitura/Gravação Objetos

- Um objeto é gravado usando o método **writeObject()** de **ObjectOutputStream**
- Um objeto é lido usando o método **readObject()** de **ObjectInputStream**
- Se uma classe serializada for alterada, um objeto gravado (serializado) com a versão antiga da classe não pode ser lido para essa nova versão – não é possível recuperar arquivos gravados com a versão antiga

Leitura/Gravação Objetos

- Escrita de objetos

```
1 File arquivo = new File("arquivo.objs");
2 ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(↩
      arquivo));
3 Data d1 = new Data(1,1,2005);
4 Data d2 = new Data(1,1,2006);
5 out.writeObject(d1);
6 out.writeObject(d2);
7 out.close();
```

- Leitura de objetos

```
1 File arquivo = new File("arquivo.objs");
2 ObjectInputStream in = new ObjectInputStream(new FileInputStream(↩
      arquivo));
3 Data d1 = (Data)in.readObject();
4 Data d2 = (Data)in.readObject();
5 in.close();
```

Leitura/Gravação Arquivos Compactados

- Usando **java.util.zip** é possível armazenar dados de forma compactada, mantendo a estrutura dos arquivos e diretórios
  - Maior eficiência para E/S
- Usa-se a classe Zip, ZipEntry, ZipFile, ZipInputStream, etc.

Leitura/Gravação Arquivos Compactados

- Leitura de arquivos compactados

```
1  ZipFile zip = new ZipFile("arquivo.zip");
2  ZipEntry entry = zip.getEntry("arquivo_interno.txt");
3  BufferedReader in = new BufferedReader(new
4  InputStreamReader(zip.getInputStream(entry));
5
6  //processa o arquivo para leitura
7  //...
8
9  zip.close();
```

- Escrita de arquivos compactados

```
1  FileOutputStream dest = new FileOutputStream("arquivo.zip");
2  ZipOutputStream zout = new ZipOutputStream(new
3  BufferedOutputStream(dest));
4
5
6  ZipEntry entry = new ZipEntry("arquivo_interno.txt");
7  zout.putNextEntry(entry);
8
9  String conteudo = ......;
10 zout.write(conteudo.getBytes(), 0, conteudo.length());
11 zout.flush();
12 zout.finish();
13 zout.close();
```