ARQUIVOS E FLUXOS

CONCEITOS BÁSICOS

BIBLIOGRAFIA

- HORSTMANN, Cay. **Conceitos de Computação com o Essencial de Java.** 3ª. Edição. São Paulo: Bookman, 2003.
  - Capítulo 15: Fluxos
- DEITEL, H. M., DEITEL, P.J. **Java – Como Programar.** 6ª ed. São Paulo: Pearson, 2005.
  - Capítulo 14: Arquivos e Fluxos

AGENDA

1. Introdução
2. Pacote java.io e a Classe File
3. Fluxos de Entrada e Saída
4. Arquivos e Fluxos
5. Entrada e Saída de Caracteres
6. Entrada e Saída de Bytes

INTRODUÇÃO

- Poucas aplicações são funcionais apenas com dados transientes;
- A grande maioria precisa armazenar informações em mídia de longa duração para recuperá-las tempos depois;
- Java trabalha com fluxos de dados, representando acesso a fontes como:
  - Memória
  - Arquivos
  - Rede
  - Etc.

INTRODUÇÃO

- Dispositivos de I/O
  - Discos (arquivos)
  - Console
  - Conexão de rede
- Formas de acesso
  - Seqüencial
  - Randômico
  - Binário
  - Por Linhas
  - Por Palavras
- O sistema de I/O associa uma forma de acesso a um dispositivo.

INTRODUÇÃO

- O pacote **java.io** oferece abstrações que permitem ao programador lidar com arquivos, diretórios e seus dados de uma maneira independente de plataforma;
- Oferece ainda recursos para facilitar a manipulação de dados durante o processo de leitura ou gravação:
  - Bytes sem tratamento
  - Caracteres Unicode
  - Dados filtrados de acordo com certo critério
  - Dados otimizados em buffers
  - Leitura/gravação automática de objetos

INTRODUÇÃO

- Diferentes sistemas operacionais representam arquivos e trilhas (*paths*) de diferentes formas:
  - *C:\Documents and Settings\User\Arquivo.txt*
  - */home/User/Arquivo.txt*
- Java utiliza a classe ***java.io.File***, abstraindo esta representação e provendo portabilidade;
- No Windows:
  - `File f = new File("C:\\pasta\\arq.txt");`
  - `File f = new File("C:/pasta/arq.txt");`
- No Linux/Unix:
  - `File f = new File("/pasta/arq.txt");`

A CLASSE java.io.File

- Usada para representar o sistema de arquivos:
  - É apenas uma abstração: a existência de um objeto File não significa a existência de um arquivo ou diretório.
  - Contém métodos para testar a existência de arquivos, para definir permissões (nos sistemas operacionais onde for aplicável), para apagar arquivos, criar diretórios, listar o conteúdo de diretórios, etc.
- Pode representar arquivos ou diretórios:
  - `File a1 = new File("arq1.txt");`
  - `File a2 = new File("C:/pasta",`

A CLASSE java.io.File

- Alguns métodos:
  - String getAbsolutPath() : retorna o endereço absoluto
  - String getParent(): retorna o diretório (objeto File) pai
  - boolean exists() : verifica se o diretório ou arquivo existe
  - boolean isFile() : verifica se é arquivo
  - boolean isDirectory() : verifica se é diretório
  - boolean delete(): tenta apagar o diretório ou arquivo
  - long length(): retorna o tamanho do arquivo em bytes
  - boolean mkdir(): cria um diretório com o nome do arquivo
  - String[] list(): retorna lista de arquivos contido no diretório
  - Etc.

EXERCÍCIO 01

Utilizando a classe java.io.File, faça um programa que:

- Crie um diretório;
- Crie um arquivo;
- Liste arquivos e diretórios existentes;
- Para cada um mostrar:
  - É diretório ou arquivo;
  - Nome;
  - Endereço;
  - Diretório pai;
  - Data da última modificação;
  - Tamanho;
- Remover um arquivo específico.

CAIXAS DE DIÁLOGO DE ARQUIVO

- O diálogo ***JFileChooser*** (java.swing) permite aos usuários selecionar um arquivo, navegando pelos diretórios.
- Um objeto ***File*** descreve um arquivo ou diretório.
- Podemos passar um objeto ***File*** ao construtor de um leitor de arquivos, um gravador ou fluxo.

EXEMPLO

```
JFileChooser chooser = new JFileChooser();
chooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
File file = null;
if (chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
   file = chooser.getSelectedFile();
   JOptionPane.showMessageDialog(null, file.getAbsolutePath());
}
```

EXERCÍCIO 02

Utilizando a classe java.io.File, faça um programa que:

- 1 - Mostre a hierarquia de diretórios e arquivos de um dado diretório.
- 2 - Mostre o tamanho dos arquivos contidos em cada diretório dentro de um dado diretório.

FLUXOS DE ENTRADA E SAÍDA

- Há várias fontes de onde se deseja ler ou destinos para onde se deseja gravar ou enviar dados:
  - Arquivos
  - Conexões de rede
  - Console (teclado/tela)
  - Memória
- Há várias formas diferentes de ler/escrever dados:
  - Seqüencialmente / aleatoriamente
  - Como bytes / como caracteres
  - Linha por linha / palavra por palavra, ...
- APIs Java para I/O oferecem objetos que abstraem fontes/destinos e fluxos de bytes e caracteres.

FLUXOS DE ENTRADA E SAÍDA

FLUXOS DE ENTRADA E SAÍDA

- UNICODE fornece um número único para cada caractere.
- Não depende de:
  - Plataforma
  - Programa
  - Linguagem

FLUXOS DE ENTRADA E SAÍDA

- Java cria 3 objetos de fluxos que são associados com dispositivos:
  - System.in (objeto de fluxo entrada padrão – ex. teclado);
  - System.out (objeto de fluxo saída padrão – ex. tela);
  - System.err (objeto de fluxo de erro padrão – ex. tela);

ARQUIVOS E FLUXOS

- Java trata os arquivos como um fluxo seqüencial de dados;
- Cada arquivo acaba com um marcador de fim de arquivo ou em um número específico de bytes registrado em uma estrutura administrativa de dados mantido pelo sistema;
- Um programa Java abre um arquivo através da criação de um objeto e a associação de um fluxo a este objeto.
- Também pode associar os fluxo com dispositivos;

ARQUIVOS E FLUXOS

- Até Java 1.4, I/O era feita por:
  - Fluxos (*streams*): subclasses de InputStream e OutputStream para leitura/escrita byte a byte;
  - Leitores (*readers*) e gravadores (*writers*): subclasses de Reader e Writer para leitura/escrita caractere a caractere (padrão Unicode).
- A partir do Java 5:
  - Foi criada a classe java.util.Scanner para facilitar a leitura;
  - Foram adicionados métodos à classe PrintWriter para facilitar a escrita (ex.: printf());

ARQUIVOS E FLUXOS

- Cria-se o fluxo, leitor, escritor ou *scanner* e este estará aberto automaticamente;
- Utiliza-se operações de leitura e escrita:
  - Operações de leitura podem bloquear o processo no caso dos dados não estarem disponíveis;
- Fecha-se o fluxo, leitor, escritor ou *scanner*:
  - A omissão da chamada ao método close() pode provocar desperdício de recursos ou leitura/escrita incompleta.

ARQUIVOS DE FLUXOS

- A hierarquia de classes de I/O possui mais de 40 classes, divididas em:
  - Leitores (*readers*);
  - Escritores/Gravadores (*writers*);
  - Fluxos de entrada (*input streams*);
  - Fluxos de saída (*output streams*);
  - Arquivo de acesso aleatório (*random access file*).
- Classes podem indicar a mídia de I/O ou a forma de manipulação dos dados;
- Podem (devem) ser combinadas para atingirmos o resultado desejado.

ARQUIVOS E FLUXOS

- Há duas maneiras fundamentalmente diferentes de armazenar dados:
  - formato *texto*
  - formato *binário*
- No formato texto:
  - Itens de dados são representados em uma forma legível ao homem
  - Uma seqüência de *caracteres*
  - Um *char* e composto por 16 bits (65.536 valores)
- Na forma binária:
  - Itens são representados em *bytes*
  - Um *byte* é composto de 8 bits (256 valores)

ARQUIVOS E FLUXOS

- Informações em formato texto:
  - Seqüência de caracteres.
  - Classes *Reader* e *Writer* e suas subclasses.
- Informações em formato binário:
  - Uma seqüência de bytes.
  - Classes *InputStream* e *OutputStream* e suas subclasses.

ARQUIVOS E FLUXOS

- Par ler e gravar arquivos:
  - *FileReader*
  - *FileWriter*
  - *FileInputStream*
  - *FileOutputStream*

EXEMPLOS

```
FileReader reader =
  new FileReader("input.txt");

FileInputStream inputStream =
  new FileInputStream("input.dat");

FileWriter writer =
  new FileWriter ("output.txt");

FileOutputStream outputStream =
  new FileOutputStream("output.dat");
```


ARQUIVOS E FLUXOS

- *Reader*, *FileReader* e *FileInputStream* têm o método *read()* que retorna um inteiro:
  - -1, se o final do arquivo for atingido.
  - outro valor *int*, o qual precisamos converter para *char* ou *byte*, respectivamente.
- *Writer, FileWriter* e *FileOutputStream* têm um método *write()* para gravar um único *caractere* ou *byte*, respectivamente.

EXEMPLO

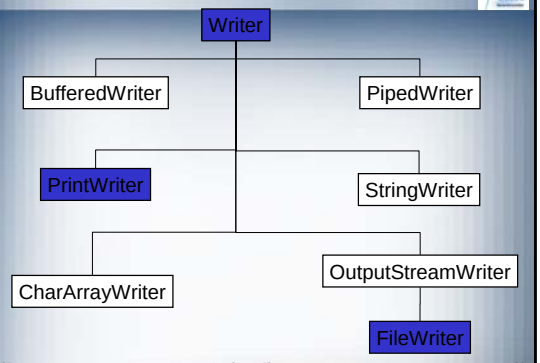
```
Reader reader = …;
int next = reader.read();
char c;
if (next != -1) c = (char) next;

InputStream input = …;
int next = input.read();
byte b;
if (next != -1) b = (byte) next;
```


ARQUIVOS E FLUXOS

- Fluxos, leitores e gravadores básicos só conseguem processar bytes ou caracteres individualmente.
- Precisamos combiná-los com outras classes para processar linhas de texto ou objetos inteiros.

SAÍDA DE CARACTERES



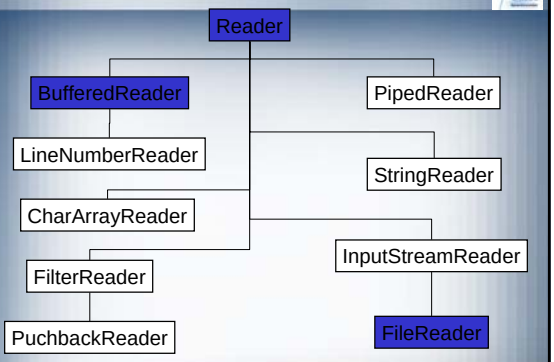
SAÍDA DE CARACTERS

- Usamos a classe ***PrintWriter*** e os métodos *print()* e *println()*.
  - Usam o método *toString()* para converter objetos para strings.
  - Quebram os strings em caracteres individuais.
  - Fornecem cada caractere para o objeto *FileWriter* através de seu método *write()*.

SAÍDA DE CARACTERES

```
File file = new File("arquivo.txt");
try {
  FileWriter writer = new FileWriter(file);
  PrintWriter output = new PrintWriter(writer);
  output.println("Unisul");
  output.println("Curso Sequencial");
  output.println("Tópicos Especiais I");
  output.println(10);
  output.flush();
  writer.close();
} catch (IOException ioe) {
  System.out.println(ioe);
}
```


ENTRADA DE CARACTERES



ENTRADA DE CARACTERES

- Usamos a classe ***BufferedReader*** e o método *readLine()*.
  - Vai chamando o método *read()* do objeto leitor que fornecemos ao construtor, até que ele tenha coletado uma linha inteira.
  - Retorna essa linha.
  - Quando toda a entrada tiver sido lida, o método *readLine()* retorna *null*.

ENTRADA DE CARACTERES

```
File file = new File("arquivo.txt");
try {
  FileReader reader = new FileReader(file);
  BufferedReader input = new BufferedReader(reader);
  String line;
  while ((line = input.readLine()) != null) {
    System.out.println(line);
  }
  input.close();
} catch (IOException ioe) {
   System.out.println(ioe);
}
```


ENTRADA E SAÍDA DE CARACTERES

- Reader
  - Classe abstrata para lidar com fluxos de caracteres de entrada.
  - método read() lê um caractere por vez.
- Writer
  - Classe abstrata para lidar com fluxos de caracteres de saída.
  - método write() grava um caractere por vez.
- Principais implementações
  - Destinos: FileWriter (arquivo), CharArrayWriter (memória), PipedWriter (pipe) e StringWriter (memória).
  - Processamento de Saída: FilterWriter (abstract), BufferedWriter, OutputStreamWriter (conversor de bytes para chars), PrintWriter.
  - Fontes: FileReader (arquivo), CharArrayReader (memória), PipedReader (pipe) e StringReader (memória).
  - Processamento de Entrada: FilterReader (abstract), BufferedReader, InputStreamReader (conversor bytes p/ chars), LineNumberReader.

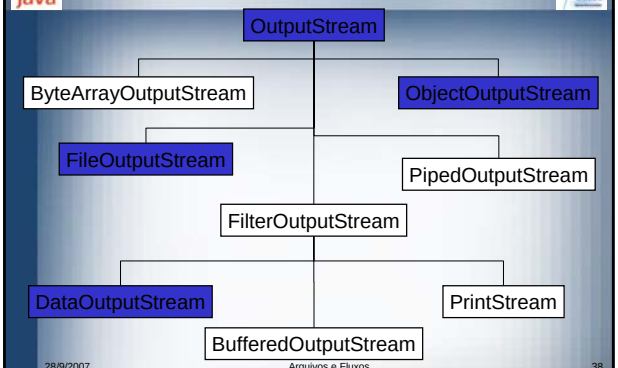
ENTRADA E SAÍDA DE CARACTERES

- Principais métodos de Reader
  - int read(): lê um char (ineficiente)
  - int read(char[] buffer): coloca chars lidos no vetor passado como parâmetro e retorna quantidade lida
  - int read(char[] buffer, int offset, int length): idem
  - void close(): fecha o stream
  - int available(): número de chars que há para ler agora
- Métodos de Writer
  - void write(int c): grava um char (ineficiente)
  - void write(char[] buffer)
  - void write(char[] buffer, int offset, int length)
  - void close(): fecha o stream (essencial)
  - void flush(): esvazia o buffer

EXERCÍCIO 03

1. Utilizando as classes vistas até agora, copie e conteúdo de um arquivo para outro. Peça para o usuário definir o arquivo de origem e o arquivo de destino.
2. Utilizando as classes vistas até agora, conte o número de linhas palavras e caracteres de um arquivo. Peça para o usuário escolher o arquivo.

FLUXO DE SAÍDA



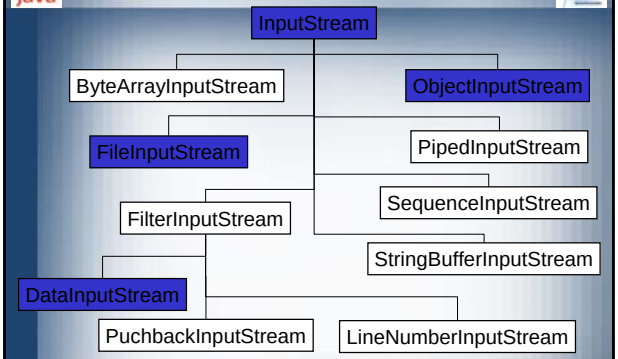
FLUXO DE SAÍDA

- Usamos a classe **DataOutputStream** e seus diversos métodos para fazer a gravação de tipos primitivos em arquivos binários:
  - writeInt(int v)
  - writeDouble(double v)
  - writeBoolean(boolean v)
  - Etc.

FLUXO DE SAÍDA

```
File file = new File("arquivo2.dat");
try {
  FileOutputStream writer = new FileOutputStream(file);
  DataOutputStream output = new DataOutputStream(writer);
  output.writeDouble(39.3);
  output.writeBoolean(true);
  output.writeInt(10000);
  writer.flush();
  writer.close();
} catch (IOException ioe) {
  System.out.println(ioe);
}
```


FLUXO DE ENTRADA



FLUXO DE ENTRADA

- Usamos a classe **DataInputStream** e seus diversos métodos para fazer a leitura de tipos primitivos de arquivos binários.
  - readInt()
  - readDouble()
  - readeBoolean()
  - Etc.

FLUXO DE ENTRADA

```
File file = new File("arquivo2.dat");
try {
  FileInputStream reader = new FileInputStream(file);
  DataInputStream input = new DataInputStream(reader);
  double d = input.readDouble();
   System.out.println(d);
  boolean b = input.readBoolean();
  System.out.println(b);
  int i = input.readInt();
  System.out.println(i);
} catch (EOFException eofe) {
  System.out.println("Final do arquivo");
} catch (IOException ioe) {
   System.out.println(ioe);
}
```


ENTRADA E SAÍDA DE BYTES

- InputStream
  - Classe genérica (abstrata) para lidar com fluxos de bytes de entrada e nós de fonte (dados para leitura).
  - Método principal: read()
- OutputStream
  - Classe genérica (abstrata) para lidar com fluxos de bytes de saída e nós de destino (dados para gravação).
  - Método principal: write()
- Principais implementações
  - Fontes: FileInputStream (arquivo), ByteArrayInputStream (memória) e PipedInputStream (pipe).
  - Processamento de Entrada: FilterInputStream (abstract) e subclasses.
  - Destinos: FileOutputStream (arquivo), ByteArrayOutputStream (memória) e PipedOutputStream (pipe).
  - Processamento de Saída: FilterOutputStream (abstract) e subclasses.

ENTRADA E SAÍDA DE BYTES

- Principais Métodos de InputStream
  - int read(byte[] buffer): Coloca bytes lidos no vetor passado como parâmetro e retorna quantidade lida ou -1 quando atinge o final do stream;
  - int read(): Lê um byte e retorna o byte lido ou -1 quando atinge o final do stream (ineficiente);
  - int read(byte[] buffer, int offset, int length): Lê uma quantidade de bytes e armazena no vetor a partir da posição inicial dada;
  - void close(): Fecha o stream;
  - int available(): Número de bytes que há para ler agora.

ENTRADA E SAÍDA DE BYTES

- Principais Métodos de OutputStream
  - void write(int c): Grava um byte;
  - void write(byte[] buffer);
  - void write(byte[] buffer, int offset, int length);
  - void close(): Fecha o stream (essencial);
  - void flush(): Esvazia o buffer;

EXERCÍCIO 04

- Escreva um programa que criptografe um arquivo:
  - A chave deve ser um número entre 1 e 25 que indica o deslocamento a ser usado para criptografar cada letra.
  - Para decodificar, simplesmente utilize o inverso da chave criptográfica.
  - Utilize arquivos binários.