ARQUIVO TEXTO

MICHELLE NERY NASCIMENTO
PUC MINAS
ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

INTRODUÇÃO

- Programadores utilizam arquivos para armazenar dados a longo prazo.
- Dados armazenados em arquivos são chamados de persistentes:
  - eles existem mesmo depois que os programas que os criaram tenham terminado.
- O termo fluxo se refere a dados que são lidos ou gravados em um arquivo.

HIERARQUIA DE DADOS

- Bit
  - menor item de dados em um computador
    - assume valor 0 ou 1
  - inadequado para leitor humano
  - usamos caracteres (dígitos, símbolos, letras)
    - representados no computador como padrões de 0's e 1's
    - em Java: caracteres Unicode compostos de dois bytes

HIERARQUIA DE DADOS

- Byte
  - Grupo de 8 bits;
  - 2 bytes (16 bits) são usados para representar um caractere Unicode;
- Campo
  - Grupo de caracteres com significado;
  - Exemplo: endereço de funcionário;
- Registro
  - Grupo de campos relacionados;
  - Exemplo: registro de um funcionário.

HIERARQUIA DE DADOS

- Arquivo
  - Grupo de registros relacionados;
  - Exemplo: informações sobre muitos funcionários;

- Banco de Dados
  - Grupo de arquivos relacionados;
  - Exemplo: arquivo de folha de pagamento, arquivo de contas a receber, arquivos de contas a pagar, etc.

Exemplo

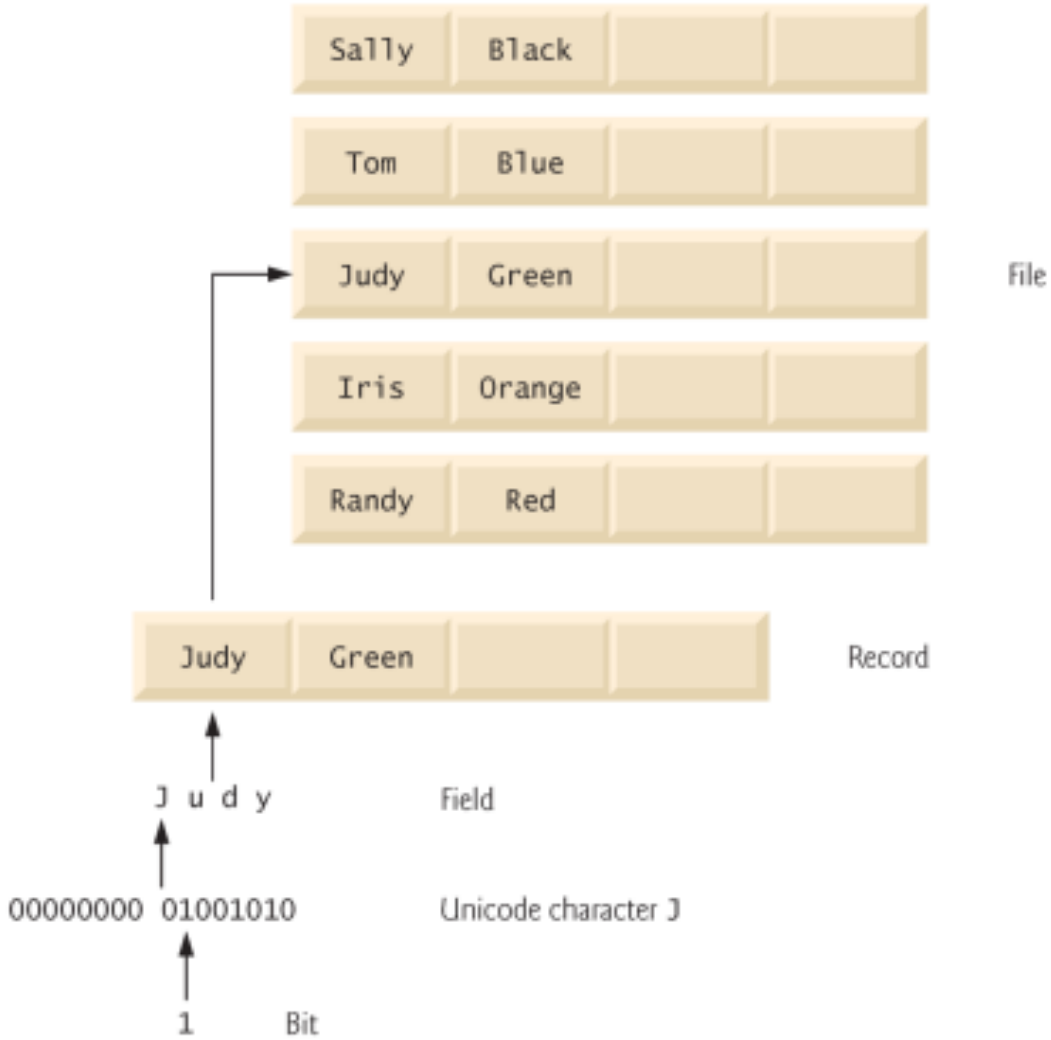

Fig. 17.1 | Data hierarchy.

ARQUIVOS E FLUXOS

- Java vê cada arquivo como um fluxo sequencial de bytes;
  - geralmente terminam com uma marca de final de arquivo ou um código especial. (Um programa Java simplesmente recebe uma indicação do S.O. quando chega ao fim do arquivo.)

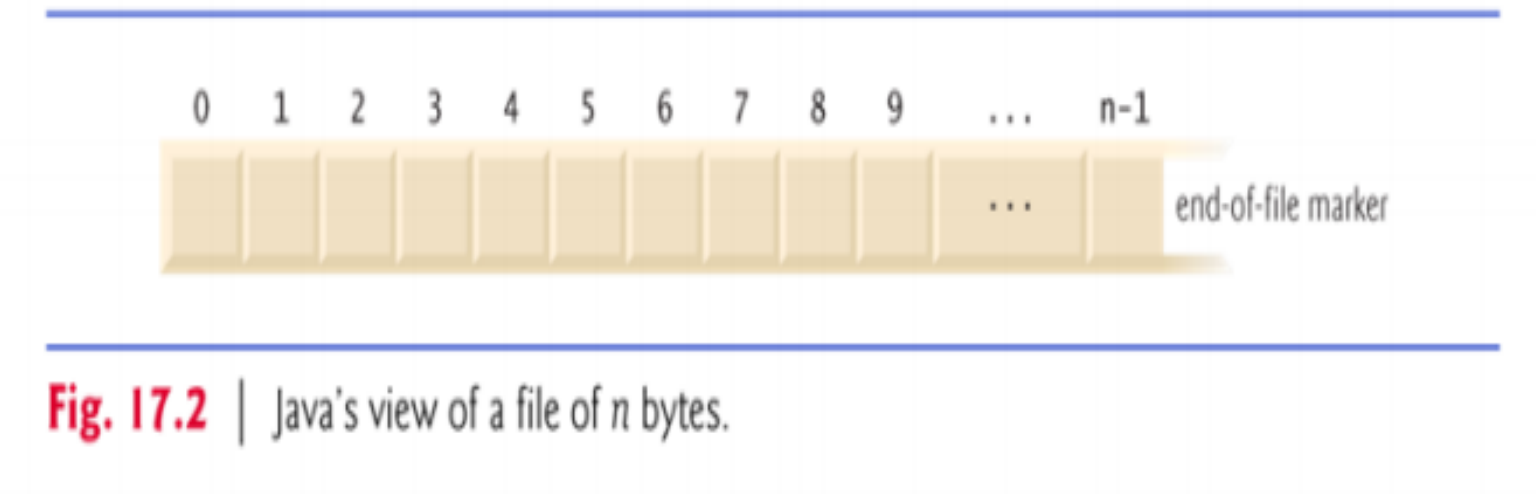

Fig. 17.2 | Java's view of a file of *n* bytes.

  - Fluxos de arquivos podem ser utilizados para entrada e saída de dados como caracteres ou bytes.

TIPOS DE ARQUIVO

- Arquivos de texto:
  - criados com base em fluxos de caracteres
  - podem ser lidos por editores de texto

- Arquivos binários
  - criados com base em fluxos de bytes
  - lidos por um programa que converte os dados em um formato legível por humanos

ARQUIVO TEXTO

- Arquivos texto em Java são criados com base nos fluxos de caracteres.
- Logo, arquivos textos são sequências de caracteres Unicode.
- Geralmente, a extensão utilizada em arquivos textos é o formato .txt, que é uma extensão que consiste pouquíssimo tipo de formatação.

CLASSE FILE

- import java.io.File
- Esta classe não consegue manipular o conteúdo de um arquivo por meio de leituras e gravações.
- Ela fornece meios de associar uma variável a um arquivo físico, ou seja, cria um caminho abstrato entre o programa e o arquivo de dados.
- Útil para recuperar informações sobre arquivos e diretórios em disco (Não abre nem processa arquivos)
- É utilizada com objetos de outras classes do pacote java.io para especificar arquivos ou diretórios a manipular.

CLASSE FILE

- Verificando se o objeto faz referência a um arquivo ou diretório(pasta) existente.

```
...
File arq = new File ("c:\\exemplo\\teste\\dados.txt);
if (arq.exists())
    System.out.println("Arquivo ou diretório existente");
else
    System.out.println("Caminho abstrato não existe fisicamente");
...
```

CLASSE FILE

- Verificando se o objeto faz referência a um diretório(pasta).

```
...
File arq = new File ("c:\\exemplo\\teste\\dados.txt);
if (arq.isDirectory())
    System.out.println("O objeto arq faz referência a um diretório");
else
    System.out.println("O objeto arq não faz referência a um diretório");
...
```

CLASSE FILE

- Verificando se o objeto faz referência a um arquivo.

```
. . .
File arq = new File (“c:\\exemplo\\teste\\dados.txt);
if (arq.isFile())
    System.out.println(“O objeto arq faz referência a um arquivo existente”);
else
    System.out.println(“O objeto arq faz referência a um arquivo inexistente”);
. . .
```

CLASSE FILE

- Criando um arquivo vazio.

```
. . .
File arq = new File (“c:\\exemplo\\teste\\dados.txt);
if (!arq.isFile())
    arq.createNewFile();
    System.out.println(“Novo arquivo vazio foi criado”);
else
    System.out.println(“O objeto arq faz referência a um arquivo existente.”);
. . .
```

CLASSE FILE

- Verificando se o objeto faz referência a um arquivo que pode ser lido.

```
. . .
File arq = new File (“c:\\exemplo\\teste\\dados.txt);
if (arq.canRead())
    System.out.println(“O arquivo pode ser lido”);
else
    System.out.println(“O arquivo não pode ser lido”);
. . .
```

CLASSE FILE

- Verificando se o objeto faz referência a um arquivo que pode receber gravações.

```
. . .
File arq = new File (“c:\\exemplo\\teste\\dados.txt);
if (arq.canWrite())
    System.out.println(“O arquivo pode receber gravações”);
else
    System.out.println(“O arquivo não pode receber gravações”);
. . .
```

CLASSE FILE

- Criando um novo diretório a partir do diretório corrente.

```
. . .
File arq = new File (“exemplo”);
if (arq.mkdir())
    System.out.println(“Diretório criado com sucesso”);
else
    System.out.println(“Erro na criação do diretório”);
. . .
```

CLASSE FILE

- Criando uma hierarquia de diretórios(pastas).

```
. . .
File arq = new File (“c:\\exemplo\\teste\\prova”);
if (arq.mkdirs())
    System.out.println(“Diretório criado com sucesso”);
else
    System.out.println(“Erro na criação do diretório”);
. . .
```

CLASSE FILE

- Apagando um arquivo ou diretório.

```
. . .
File arq = new File (“c:\\exemplo\\teste\\dados.txt”);
if (arq.delete())
    System.out.println(“Exclusão realizada com sucesso”);
else
    System.out.println(“Erro na exclusão”);
. . .
```

CLASSE FILE

- Descobrindo o tamanho de um arquivo em bytes (0 se for um diretório)

```
. . .
File arq = new File (“c:\\exemplo\\teste\\dados.txt”);
long tamanho;
tamanho = arq.length();
System.out.println(“Erro na alteração”);
. . .
```

CLASSE FILE

- Descobrindo a hora da última atualização feita no arquivo

```
. . .
File arq = new File (“c:\\exemplo\\teste\\dados.txt”);
long atualiza;
atualiza= arq.lastModified();
System.out.println(“Última atualização ocorreu em:”, atualiza”);
. . .
```

Obs.: o valor retornado é long que representa a quantidade de milissegundos existentes desde janeiro de 1970 às 00:00:00 até o momento da última atualização.

CLASSE FILE

- Descobrindo a hora da última atualização feita no arquivo

```
. . .
File arq = new File (“c:\\exemplo\\teste\\dados.txt”);
long atualiza;
atualiza= arq.lastModified();
System.out.println(“Última atualização ocorreu em:”, atualiza”);
. . .
```

Obs.: o valor retornado é long que representa a quantidade de milissegundos existentes desde janeiro de 1970 às 00:00:00 até o momento da última atualização.

CLASSE FILE : OUTROS MÉTODOS

- getAbsolutePath() - retorna uma string com o caminho absoluto do arquivo ou diretório.
- getName() - retorna uma string com o nome do arquivo ou diretório.
- getParent() - retorna uma string com o diretório-pai do arquivo ou diretório.

```java
// Fig. 17.4: FileDemonstration.java
// File class used to obtain file and directory information.
import java.io.File;
import java.util.Scanner;

public class FileDemonstration
{
   public static void main( String[] args )
   {
      Scanner input = new Scanner( System.in );

      System.out.print( "Enter file or directory name: " );
      analyzePath( input.nextLine() );
   } // end main

   // display information about file user specifies
   public static void analyzePath( String path )
   {
      // create File object based on user input
      File name = new File( path );
```

Associates a file or directory with a `File` object.

```
if ( name.exists() ) // if name exists, output information about it
{
   // display file (or directory) information
   System.out.printf(
      "%s%s\n%s\n%s\n%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s",
      name.getName(), " exists",
      ( name.isFile() ? "is a file" : "is not a file" ),
      ( name.isDirectory() ? "is a directory" :
         "is not a directory" ),
      ( name.isAbsolute() ? "is absolute path" :
         "is not absolute path" ), "Last modified: ",
      name.lastModified(), "Length: ", name.length(),
      "Path: ", name.getPath(), "Absolute path: ",
      name.getAbsolutePath(), "Parent: ", name.getParent() );
```

Determines if the file or directory exists.

```
   }
```

```
      else // not file or directory, output error message
      {
         System.out.printf( "%s %s", path, "does not exist." );
      } // end else
   } // end method analyzePath
} // end class FileDemonstration
```

```
Enter file or directory name: E:\Program Files\Java\jdk1.6.0_11\demo\jfc
jfc exists
is not a file
is a directory
is absolute path
Last modified: 1228404395024
Length: 4096
Path: E:\Program Files\Java\jdk1.6.0_11\demo\jfc
Absolute path: E:\Program Files\Java\jdk1.6.0_11\demo\jfc
Parent: E:\Program Files\Java\jdk1.6.0_11\demo
```

ARQUIVO TEXTO-PROCEDIMENTO GERAL

- LEITURA

```
abrir fluxo
enquanto houver dados
    ler
fechar fluxo
```

- GRAVAÇÃO

```
abrir fluxo
enquanto houver dados
    escrever
fechar fluxo
```

ARQUIVO TEXTO: GRAVAÇÃO DE CARACTERES

- Depois que o programa consegue estabelecer um caminho abstrato até o arquivo de dados, outras classes deverão ser utilizadas para gravação e leitura.

- A classe FileWriter define objetos capazes de escrever caracteres em um arquivo. Para isso, essa classe coloca à disposição vários métodos. Veremos alguns.

FILEWRITER: CRIANDO UM OBJETO ESCRITOR

- Exemplo 1:
  - Inicialmente foi criado o objeto arq, da classe File. Depois, foi criado o objeto escritor, vinculando-se a arq, ou seja, escritor conseguirá gravar caracteres no arquivo arq.

```
File arq = new File ("c:\\exemplo\\teste\\dados.txt");
FileWriter escritor = new FileWriter (arq);
```

FILEWRITER: CRIANDO UM OBJETO ESCRITOR

- Exemplo 2:
  - Inicialmente foi criado o objeto arq, da classe File. Depois, foi criado o objeto escritor, vinculando-se a arq, ou seja, escritor conseguirá gravar caracteres no arquivo arq.
  - O segundo parâmetro, true, quer dizer que será permitido o acréscimo de novos caracteres a um arquivo já existente. Se este parâmetro for suprimido, toda vez que ocorrer uma gravação, os dados anteriormente existentes no arquivo serão destruídos.

```
File arq = new File ("c:\\exemplo\\teste\\dados.txt");
FileWriter escritor = new FileWriter (arq, true);
```

FILEWRITER: WRITE

- Gravando um caractere em um arquivo de texto.

```
. . .
char caractere = 'x';
File arq = new File ("c:\\exemplo\\teste\\dados.txt");
FileWriter escritor = new FileWriter (arq, true);
escritor.write(caractere);
. . .
```

FILEWRITER: APPEND

- Acrescentando um caractere em um arquivo de texto já existente.

```
. . .
char carcterer = 'x';
File arq = new File ("c:\\exemplo\\teste\\dados.txt");
FileWriter escritor = new FileWriter (arq, true);
escritor.append(caracter);
. . .
```

Obs.: o método append() funciona da mesma forma que o método write() descrito no slide anterior.

FILEWRITER: CADEIA DE CARACTERES

- Gravando uma cadeia de caracteres em um arquivo de texto.

```
. . .
String cadeia;
cadeia = “exemplo de gravação”;
File arq = new File (“c:\\exemplo\\teste\\dados.txt”);
FileWriter escritor = new FileWriter (arq, true);
escritor.write(cadeia);
. . .
```

FILEWRITER: CLOSE

- Fechando o objeto de gravação no arquivo.

```
...
File arq = new File ("c:\\exemplo\\teste\\dados.txt");
FileWriter escritor = new FileWriter (arq);
...
escritor.close();
...
```

Obs.: Ao término de qualquer operação, o arquivo deve ser fechado. Um arquivo fechado não permitirá a realização de nenhuma operação nos dados.

EXEMPLO:

```
import java.io.*;
public class ExemploFileWriter {
  public static void main(String[] args) throws IOException{
      FileWriter escrita = new FileWriter("fwriter1.txt");
      String txt = "Era uma vez um gato xadrez";
      escrita.write(txt);
      escrita.close();
      System.out.println("fim");
  }
}
```

ARQUIVO TEXTO: LEITURA DE CARACTERES

- A classe FileReader define objetos capazes de ler caracteres de um arquivo. Para isso, essa classe coloca à disposição vários métodos. Veremos alguns.

FILEREADER: CRIANDO UM OBJETO LEITOR

- Exemplo 1:
  - Inicialmente foi criado o objeto arq, da classe File. Depois, foi criado o objeto leitor, vinculando-se a arq, ou seja, leitor conseguirá extrair caracteres do arquivo arq.

```
File arq = new File ("c:\\exemplo\\teste\\dados.txt");
FileReader leitor = new FileReader (arq);
```

FILEREADER: CRIANDO UM OBJETO LEITOR

- Exemplo 2:
  - Neste exemplo, o objeto leitor foi criado, vinculanso-se a um caminho especificado, ou seja, leitor conseguirá extrair caracteres do arquivo dados.txt localizado em c:\\exemplo\\teste.

```
FileReader leitor = new FileReader (“c:\\exemplo\\teste\\dados.txt”);
```

FILEWRITER: READ

- Lendo um caractere do arquivo texto.

```
. . .
char carac;
File arq = new File (“c:\\exemplo\\teste\\dados.txt”);
FileReader leitor = new FileReader (arq);
carac = (char) leitor.read();
System.out.println(“Caractere lido do arquivo texto:“ + carac);
. . .
```

Obs.: Como o método read () retorna um inteiro, é preciso converter esse dado antes de ser exibido/armazenado. O inteiro retornado é um código numérico que representa o caractere no conjunto de caracteres Unicode. Retorna -1 caso encontre final de arquivo.

FILEWRITER: READ

- Lendo uma cadeia de caractere do arquivo texto.

```
. . .
char cadeia [] = new char[5];
File arq = new File ("c:\\exemplo\\teste\\dados.txt");
FileReader leitor = new FileReader (arq);
int t = leitor.read(cadeia);
. . .
```

Inicialmente foi criado o objeto *arq*, da classe *File*. Depois, foi criado o objeto leitor, vinculando-se a *arq*, ou seja, *leitor* conseguirá ler uma cadeia de caracteres do arquivo arq de tamanho igual ao da variável *cadeia* (nesse exemplo, *cadeia* é um vetor de char com cinco posições, logo, o método *read*() lerá cinco caracteres do arquivo). A variável *t* receberá a informação de quantos caracteres realmente foram lidos. Se *t* assumir -1, significa que o fim do arquivo foi encontrado.

FILEWRITER: SKIP

- Pulando caracteres em arquivo de leitura.

```
...
File arq = new File ("c:\\exemplo\\teste\\dados.txt");
FileReader leitor = new FileReader (arq);
leitor.skip(tam);
...
```

Obs.: Note que, nesse caso, o método skip() recebeu um parâmetro: tam, que representa um valor inteiro correspondente à quantidade de caracteres que serão pulados dentro do arquivo de texto. Assim, uma leitura conseguirá capturar os caracteres a partir da posição tam + 1.

FILEWRITER: CLOSE

- Fechando o objeto de leitura de arquivo.

```
...
File arq = new File ("c:\\exemplo\\teste\\dados.txt");
FileReader leitor = new FileReader (arq);
...
leitor.close();
...
```

Obs.: Ao término de qualquer operação, o arquivo deve ser fechado. Um arquivo de leitura fechado não poderá realizar nenhuma operação no arquivo de dados.

EXEMPLO: Cópia de arquivo

```
public static void main(String[] args) throws IOException {
    File arq_entrada = new File("entrada.txt");
    File arq_saida = new File("saida.txt");
    FileReader entrada = new FileReader(arq_entrada);
    FileWriter saida = new FileWriter(arq_saida);
    int c;
    while ((c = entrada.read()) != -1)      // -1 indica final de arquivo de caracteres
        saida.write(c);
    entrada.close();
    saida.close();
}
```

EXEMPLO

```
import java.io.*;
public class ReadCaracteres{
   public static void main(String[] args) throws IOException
   {
      int i;
      File arq = new File (“c:\\exemplo\\teste\\exemplo.txt”);
      FileReader entrada = new FileReader(arq);
      while (true){
         i = entrada.read();
         if ( i == -1 ) break;
         char c = (char) i;
         System.out.print( c );
      }
   }
   System.out.println();
   entrada.close()
}
```

EXEMPLO

```
import java.util.Scanner;


class LerArquivoComScanner {
    public static void main(String[] args) throws FileNotFoundException {
        File arquivo = new File("C:\\temp\\arquivo.txt");
        Scanner sc = new Scanner(arquivo);
        while (sc.hasNextLine()) { //hasNextLine retorna true se há outra linha para ler
            System.out.println(sc.nextLine());
        }
        sc.close();
    }
}
```

EXCEÇÕES

- As instâncias de Exception, ou de qualquer outra de suas subclasses, são verificadas (checked) como, por exemplo, IOException, FileNotFoundException , etc.
  - Elas representam erros que podem ocorrer em tempo de execução, mas que não dependem da lógica do programa, em geral defeitos nos dispositivos de entrada ou saída (arquivos, rede, etc).
- O compilador exige que um método onde possam ocorrer exceções verificadas faça uma de duas coisas:
  - utilize blocos try/catch para capturar e tratar essas exceções;
  - ou declare que pode lançar essas exceções, colocando uma cláusula "throws" no seu cabeçalho, como por exemplo:
    - public static void main(String[] args) throws IOException
      - Fazendo isso, você vai indicar que aquele método vai receber a exceção e vai 'passar pra frente' ao invés de tratar.

```
private static void main(String[] args) {
    File arq = new File(C:\\TutorialArquivos\\user.txt");
    try {
        //Indicamos o arquivo que será lido
        FileReader fileReader = new FileReader(arq);
        //Criamos o objeto bufferReader que nos
        // oferece o método de leitura readLine()
        BufferedReader bufferedReader = new
BufferedReader(fileReader);
        //String que irá receber cada linha do arquivo
        String linha = "";
        //Fazemos um loop linha a linha no arquivo,
        // enquanto ele seja diferente de null.
        //O método readLine() devolve a linha na
        // posicao do loop para a variavel linha.
        while ( ( linha = bufferedReader.readLine() ) != null) {
            //Aqui imprimimos a linha
            System.out.println(linha);
        }
            //liberamos o fluxo dos objetos ou
        fechamos o arquivo
            fileReader.close();
            bufferedReader.close();
        } catch (IOException e) {
                e.printStackTrace();
        }
}
```

TRY/CATCH

- Esta exceção "IOException" é lançada pelas classes do pacote IO.
- toda vez que você for usar alguma classe dessas é possível que lance uma exceção desse tipo,
- Importante lembrarmos de usar sempre um bloco try catch ou o throws.
- No caso do bloco try catch, se acontecer algum problema no try, é deslocado para o catch.

- No exemplo, tem-se o try catch sendo utilizado com a sintaxe catch (IOException e)
  - significa que estamos declarando um objeto do tipo IOException chamado 'e',
  - na linha seguinte utilizamos um método deste objeto chamado printStackTrace(), que consegue dar o "rastro" do erro, permitindo assim a descobrir onde ocorreu a exceção…

EXERCÍCIOS

1) Desenvolva um programa que solicite a entrada de nome, cidade e estado de um determinado usuário e grave em um arquivo de texto.

2) Desenvolva um programa que exiba na tela as informações gravadas no arquivo de texto gerado no exercício 1.

3) Desenvolva um programa que acrescente mais um dado, o número de telefone, no arquivo gerado no exercício 1.

OBRIGADA!

AVISO LEGAL

- O material presente nesta apresentação foi produzido a partir de informações próprias e coletadas de documentos obtidos publicamente a partir da Internet. Este material contém ilustrações adquiridas de bancos de imagens de origem privada ou pública, não possuindo a intenção de violar qualquer direito pertencente à terceiros e sendo voltado para fins acadêmicos ou meramente ilustrativos. Portanto, os textos, fotografias, imagens, logomarcas e sons presentes nesta apresentação se encontram protegidos por direitos autorais ou outros direitos de propriedade intelectual.
- Ao usar este material, o usuário deverá respeitar todos os direitos de propriedade intelectual e industrial, os decorrentes da proteção de marcas registradas da mesma, bem como todos os direitos referentes a terceiros que por ventura estejam, ou estiveram, de alguma forma disponíveis nos slides. O simples acesso a este conteúdo não confere ao usuário qualquer direito de uso dos nomes, títulos, palavras, frases, marcas, dentre outras, que nele estejam, ou estiveram, disponíveis.
- É vedada sua utilização para finalidades comerciais, publicitárias ou qualquer outra que contrarie a realidade para o qual foi concebido. Sendo que é proibida sua reprodução, distribuição, transmissão, exibição, publicação ou divulgação, total ou parcial, dos textos, figuras, gráficos e demais conteúdos descritos anteriormente, que compõem o presente material, sem prévia e expressa autorização de seu titular, sendo permitida somente a impressão de cópias para uso acadêmico e arquivo pessoal, sem que sejam separadas as partes, permitindo dar o fiel e real entendimento de seu conteúdo e objetivo. Em hipótese alguma o usuário adquirirá quaisquer direitos sobre os mesmos.
- O usuário assume toda e qualquer responsabilidade, de caráter civil e/ou criminal, pela utilização indevida das informações, textos, gráficos, marcas, enfim, todo e qualquer direito de propriedade intelectual ou industrial deste material.