INE5603 Introdução à POO

Prof. A. G. Silva

13 de novembro de 2017

Introdução à manipulação de arquivos

Arquivos (I)

- **Fluxo (stream)** – sequência de bytes que podem ser obtidos/enviados de um arquivo ou conexão de rede ou mesmo um bloco de memória
- **Arquivo** – sequência de bytes; e, em uma abstração de nível mais alto, uma sequência de caracteres ou objetos
- Em Java, arquivos são representados por objetos. Assim, a leitura ou escrita em um arquivo implica na construção e utilização de objetos de determinadas classes encontradas no pacote **java.io.***, tais como:
  - FileInputStream, FileOutputStream
  - DataInputStream
  - ObjectInputStream,
  - SequenceInputStream
  - ...

Arquivos (II)

- Em Java, as operações de entrada e saída podem ser:
  - **baseada em bytes** – arquivo como uma sequência de bytes;
    lê bytes; escreve bytes
  - **baseada em caracteres** – arquivo como uma sequência de caracteres no formato Unicode (2 bytes por caracter);
    lê caracter; escreve caracter

Arquivos (III)

- Exemplo de **saída**:

```
import java.io.*;
public class Principal {
    public static void main(String a[]) {
        int i;
        String s = "c:/local/saida.txt";
        try {
            FileOutputStream arqSaida = new FileOutputStream(s);
            for (i=0; i<10; i++)
                arqSaida.write(i);
            arqSaida.close();
        }
        catch (IOException ex) {
            System.out.println("Problemas de abertura/escrita...");
        }
    }
}
```

Arquivos (IV)

- Exemplo de **entrada**:

```
import java.io.*;
public class Principal {
    public static void main(String arg[]) {
        int i;
        String s = "c:/local/saida.txt";
        try {
            FileInputStream arqEntrada = new FileInputStream(s);
            i = 0;
            while (i != -1) {
                i = arqEntrada.read();
                System.out.println(i);
            }
            arqEntrada.close();
        }
        catch (IOException ex) {
            System.out.println("Problemas de abertura/leitura...");
        }
    }
}
```

Arquivos (V)

- Exemplo de **entrada** e **saída**:

```
import java.io.*;
public class Principal {
    public static void main(String a[]) {
        int i;
        String s1 = "c:/local/saida.txt";
        String s2 = "c:/local/copia.txt";
        try {
            FileInputStream arqEntrada = new FileInputStream(s1);
            FileOutputStream arqSaida = new FileOutputStream(s2);
            i = 0;
            while (i != -1) {
                i = arqEntrada.read();
                if (i != -1) arqSaida.write(i);
            }
            arqSaida.close();
            arqEntrada.close();
        }
        catch (IOException ex) {
            System.out.println("Problemas abert./leit./escr...");
        }
    }
}
```

Arquivos (VI)

- As classes `FileInputStream` / `FileOutputStream` apresentam apenas métodos para leitura/escrita de um byte ou de um array de bytes.

- Se uma aplicação necessitar ler/escrever de um arquivo, valores de tipos primitivos da linguagem (`int`, `double`, `char`, ...), são necessários objetos das classes `DataInputStream` e `DataOutputStream`

Arquivos (VII)

- Exemplo de **saída**:

```
import java.io.*;
public class Principal {
    public static void main(String arg[]) {
        String s="c:/local/saida.txt";
        String Nome = "Universidade Federal de Santa Catarina";
        double orcamento = 1500456.30;
        int numeroServidores = 5000;
        char conceito = 'A' ;
        try {
            FileOutputStream arqSaida = new FileOutputStream(s);
            DataOutputStream dadosSaida = new DataOutputStream(
                arqSaida); //objeto enviado ao construtor
            dadosSaida.writeUTF(Nome);
            dadosSaida.writeDouble(orcamento);
            dadosSaida.writeInt(numeroServidores);
            dadosSaida.writeChar(conceito);
            dadosSaida.close();
        }
        catch (IOException ex) {
            System.out.println("Problemas de abertura/escrita...");
        }
    }
}
```

Arquivos (VIII)

- Exemplo de **entrada**:

```
import java.io.*;
public class Principal {
    public static void main(String arg[]) {
        String s = "c:/local/saida.txt";
        String a; double b; int c; char d;
        try {
            DataInputStream dadosEntrada = new DataInputStream(new
                FileInputStream(s));
            a = dadosEntrada.readUTF();
            b = dadosEntrada.readDouble();
            c = dadosEntrada.readInt();
            d = dadosEntrada.readChar();
            dadosEntrada.close();
            System.out.println(a+" "+b+" "+" "+c+" "+d);
        }
        catch (IOException ex) {
            System.out.println("Problemas de abertura/leitura...");
        }
    }
}
```

Arquivos (IX)

- Uma exceção `EOFException` acontece se um operação de leitura for realizada e não houver bytes a serem lidos. Tal exceção pode ser usada para controlar `EOF` (*End Of File*). Exemplo:

```
long v;
try {
    DataInputStream dadosEntrada = new DataInputStream(new
        FileInputStream(str));
    boolean ler = true;
    while (ler) {
        try {
            v = dadosEntrada.readLong();
            System.out.println(v);
        }
        catch (EOFException eof) {
            ler = false;
        }
    }
    dadosEntrada.close();
}
catch (IOException ex) {
    System.out.println("Problemas de abertura/leitura/escrita...");
}
```

Arquivos (X)

- **Saída baseada em caracteres – saída de texto**
  - Para termos uma saída na forma de texto, devemos ter um objeto da classe `PrintWriter`. Pode-se construi-lo das seguintes formas:

    ★
    ```
    PrintWriter saida = new PrintWriter ( new
    FileWriter("saida.txt") );
    ```

    ★
    ```
    PrintWriter saida = new PrintWriter ( new
    FileOutputStream("saida.txt") );
    ```

    ★
    ```
    PrintWriter saida = new PrintWriter ( new
    FileWriter("saida.txt"), true );
    /* Com o segundo argumento true tem-se um escritor com auto
    esvaziamento do buffer (autoflush), isto é, não bufferizado */
    ```

  - Para escrever no fluxo de saída usa-se os métodos `print` e println.

Arquivos (XI)

- **Entrada baseada em caracteres – entrada de texto**
  - Para termos uma entrada na forma de texto devemos ter um objeto da classe `BufferedReader`. Pode ser construído conforme a seguir:
    - ★ `BufferdReader entra= new BufferedReader ( new FileReader("entrada.txt") );`
  - Para efetuar a leitura usa-se o método `String readLine()` que lê uma linha (um string) de texto. Este método retorna `null` quando não há uma entrada a ser lida.

Arquivos (XII)

- Exemplo:

```
import java.io.*;
public class Principal {
    public static void main(String[] aa) throws Exception { //
        propaga excecao
        PrintWriter saida = new PrintWriter( new FileWriter("saida.
            txt") );
        for (int i=1; i<5 ; i++)
            saida.println(i);
        saida.close();
        int valor, soma=0;
        BufferedReader en = new BufferedReader(new FileReader("
            saida.txt"));
        String s;
        s = en.readLine();
        while (s != null) {
            valor = Integer.parseInt(s);
            soma += valor;
            s = en.readLine();
        }
        System.out.println(soma);
    }
}
```

Arquivos (XIII)

- Classe `File` (gerenciamento de arquivo):
  - Java apresenta a classe `File` que disponibiliza métodos que permitem operações tais como: ver se um arquivo existe, saber quando o arquivo foi modificado pela última vez, alterar o nome de um arquivo, etc.
  - Principais métodos:
    - ★ `public File (String str)`
      Constrói um objeto associado ao arquivo definido por str. Veja que o objetivo não é criar o arquivo str.
    - ★ `public File(String caminho, String nome)`
      Construtor. Especifica-se o caminho e o nome do arquivo.
    - ★ `public boolean canRead()`
      Indica se o arquivo pode ser lido pelo aplicativo atual.
    - ★ `public boolean canWrite()`
      Indica se o arquivo pode ser alterado.
    - ★ `public boolean delete()`
      Tenta remover o arquivo. Caso consiga retorna true.