Linguagem de Programação

Aula 16
Manipulação de arquivos

I/O

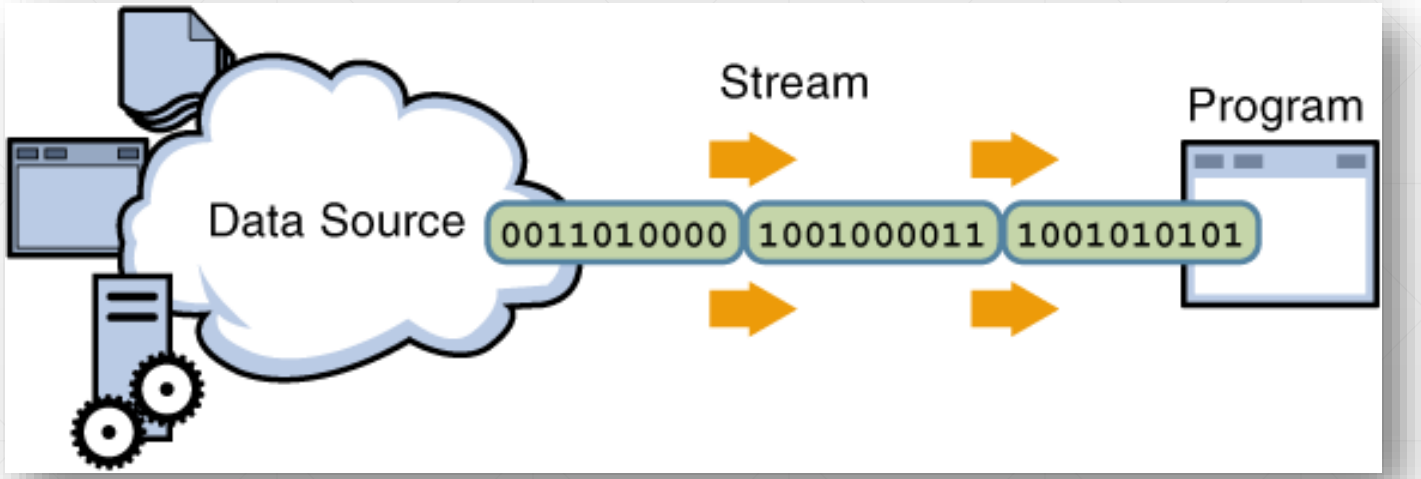
- A API Java oferece uma série de classes para realizar (IN/OUT) Entrada e Saída de arquivos;
- Classes para manipular cadeias de dados e caracteres podem ser encontradas no pacote java.io;
- Classes para manipulação de arquivos podem ser encontradas no pacote java.io.

I/O Stream

- Um Stream (cadeia ou corrente) representa a origem da entrada ou o destino da saída;
- Podem representar vários tipos de entradas e a saídas como:
  - Arquivos;
  - Dispositivos;
  - Outros programas;
  - Arranjos de memória.
- Suporta diferentes tipos de dados:
  - Bytes;
  - Tipos primitivos;
  - Caracteres;
  - Objetos.

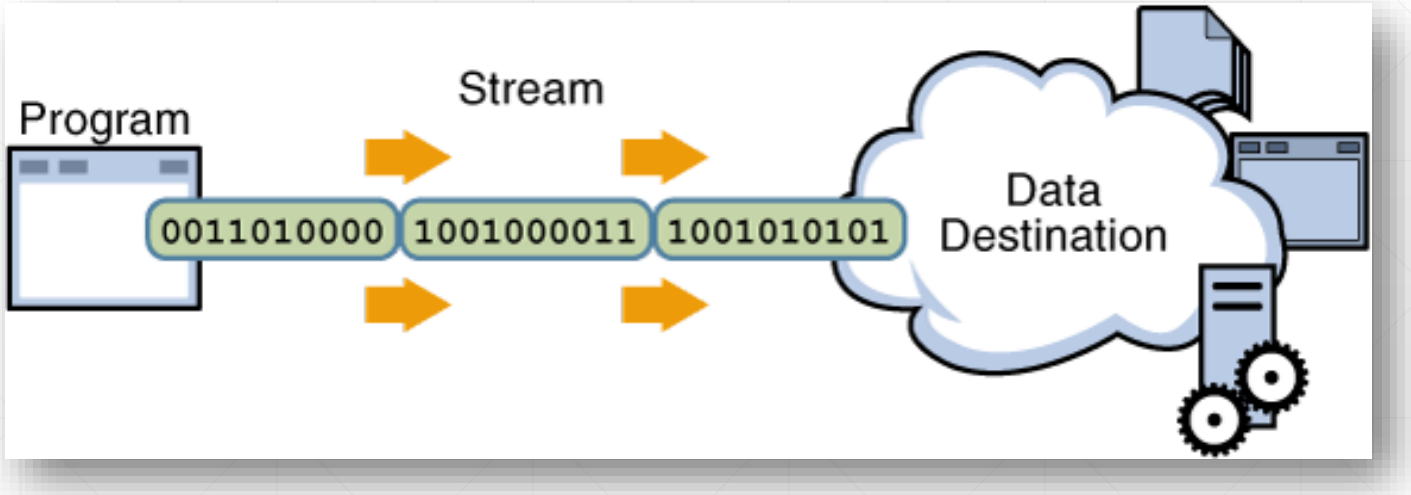
Input Stream

- Um programa utiliza um InputStream para ler dados de uma fonte de origem, um por vez.



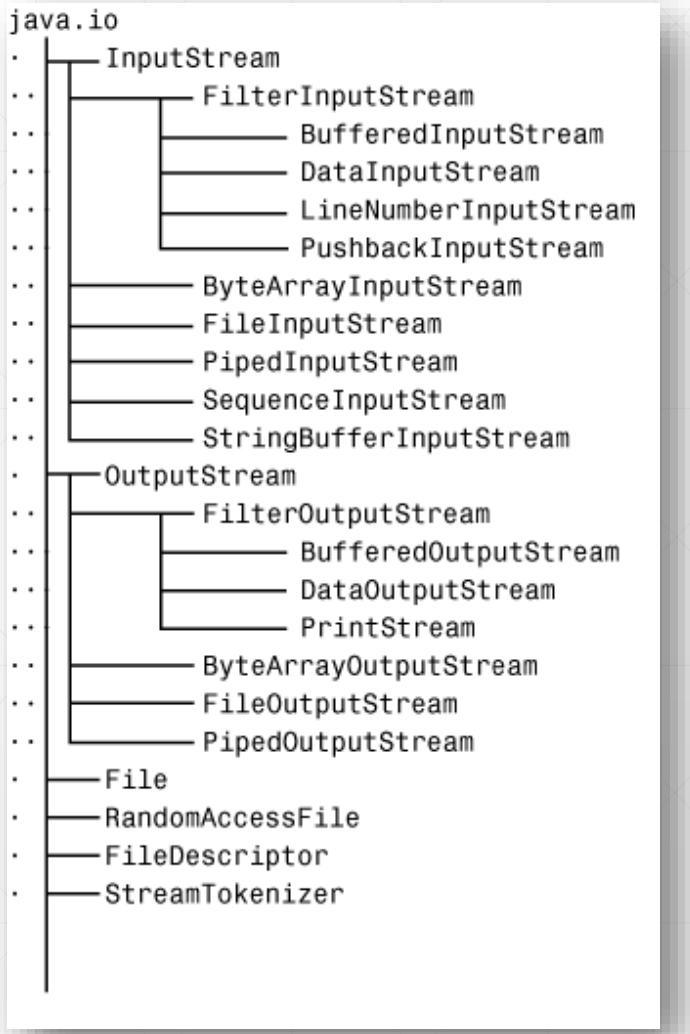
Output Stream

- Um programa utiliza um OutputStream para gravar dados em uma fonte de destino, um por vez.



ByteStreams

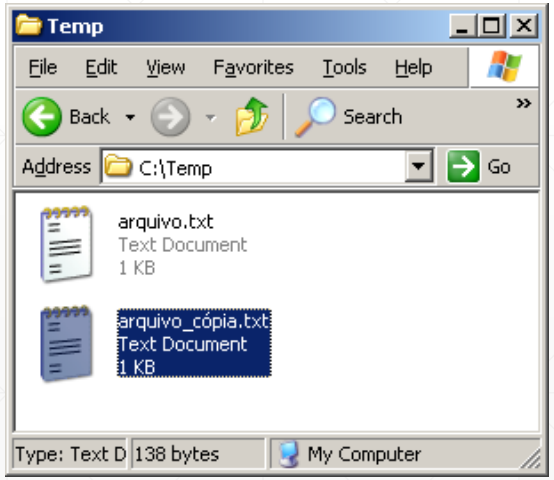
- Utilizado para ler e gravar bytes de 8 bits;
- Todas são descendentes de InputStream e OutputStream;
- No próximo slide, temos um exemplo utilizando a FileInputStream e FileOutputStream.



ByteFileStreams – Copiando um arquivo

Main.java

```
1  package jappio;
2
3  import java.io.FileInputStream;
4  import java.io.FileNotFoundException;
5  import java.io.FileOutputStream;
6  import java.io.IOException;
7
8  public class Main {
9      public static void main(String[] args) throws FileNotFoundException, IOException {
10         FileInputStream entrada = null; //declaração do strem de entrada
11         FileOutputStream saída = null;  //declaração do strem de saída
12         try {
13             //Instancia os dois streams
14             entrada = new FileInputStream("c:/TEMP/arquivo.txt");
15             saída = new FileOutputStream("c:/TEMP/arquivo_cópia.txt");
16             //Rotina que lê byte a byte da entrada, gravando-os na saída
17             int _byte;
18             while ((_byte = entrada.read()) != -1) {
19                 saída.write(_byte);
20             }
21         } finally {
22             //Fecha os streams, consolidando as leituras e gravações
23             if (entrada != null) {entrada.close();}
24             if (saída != null) {saída.close();}
25         }
26     }
27 }
```



Considerações

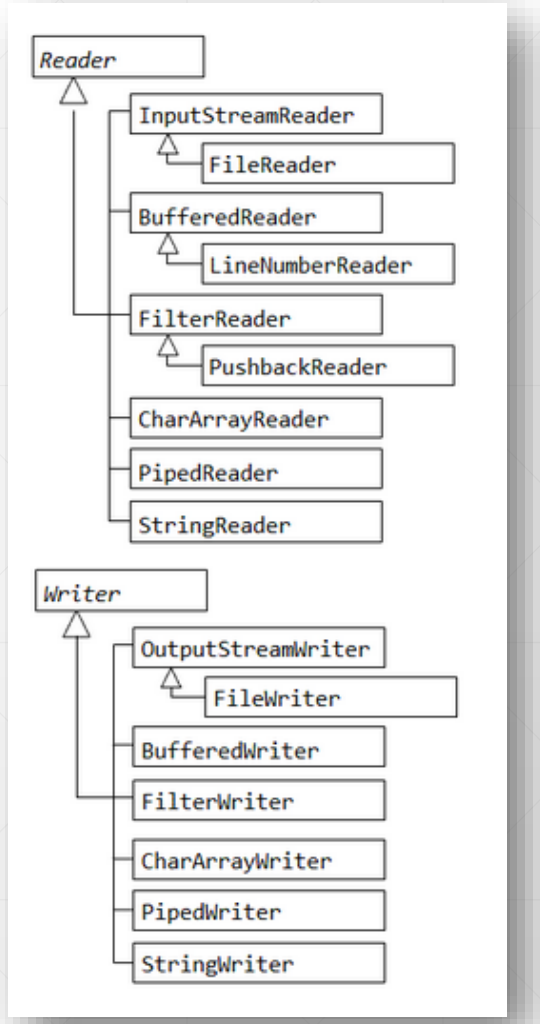
- Evidentemente a pasta e o arquivo original devem existir;
- A pasta e o arquivo podem ser criados manualmente;
- Java considera o caracter ‘/’ como sendo o ‘\’ quando utilizados em caminhos de arquivos.

Considerações

- Por quê o método read() retorna int e não byte?
  - Retornando um int ele permite que utilizemos o -1 para sinalizar que não há mais bytes para serem lidos e finalizar o loop;
- Sempre fechar o stream
  - Garantir que será executado o método close() do stream, mesmo se acontecer uma exceção evita vazamento de recursos;
- O ByteStream deve ser utilizado para ler e gravar arquivos binários, não arquivos texto.

CharacterStream

- A plataforma JAVA utiliza as convenções Unicode, ASCII;
- Todas as classes são descendentes de Reader e Writer;
- No slide a seguir, um exemplo utilizando as classes FileReader e FileWriter.

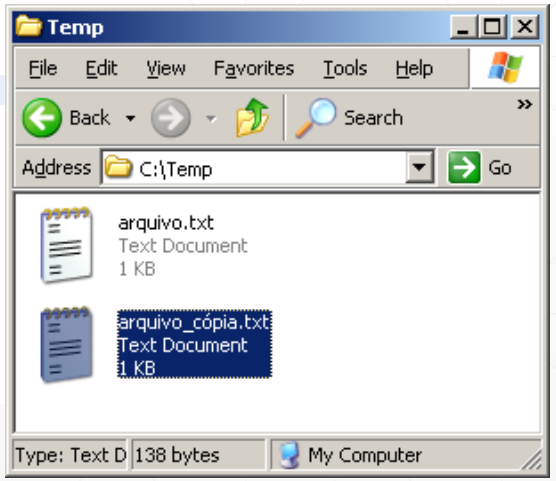


FileCharacterStreams

```java
package jappio;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws FileNotFoundException, IOException {
        FileReader entrada = null; //declaração do arquivo de entrada
        FileWriter saída = null;  //declaração do arquivo de saída
        try {
            //Instancia leitor e gravador de arquivo
            entrada = new FileReader("c:/TEMP/arquivo.txt");
            saída = new FileWriter("c:/TEMP/arquivo_cópia.txt");
            //Rotina que lê caracter a caracter da entrada, gravando-os na saída
            int _character;
            while ((_character = entrada.read()) != -1) {
                saída.write(_character);
            }
        } finally {
            //Fecha os arquivos, consolidando as leituras e gravações
            if (entrada != null) {entrada.close();}
            if (saída != null) {saída.close();}
        }
    }
}
```



Considerações

- A diferença, além do uso de classes diferentes, fica por conta do método read();
- Num ByteStrem, o read() retorna um byte de 8 bits;
- Num CharacterStream, o read() retorna um character de 16 bits.

I/O linha-a-linha

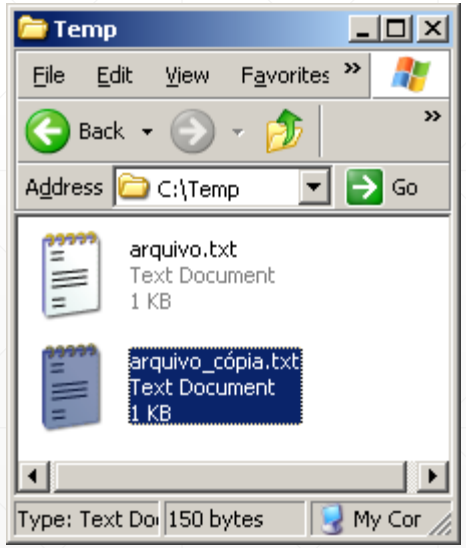
- É mais fácil manipular linhas inteiras do que caracteres;
- Um terminador de linha pode ser:
  - Carriage-return + Line Feed: “\r\n”;
  - Carriage-return: “\r”;
  - Line Feed: “\n”.
- No exemplo a seguir, utilizamos as classes BufferedReader e PrintWriter para copiar um arquivo texto linha-a-linha e não character-a-caracter.

BufferedReader & PrintWriter

```java
package jappio;

import java.io.FileReader;
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.IOException;

public class Main {
    public static void main(String[] args) throws FileNotFoundException, IOException {
        BufferedReader entrada = null; //declaração do arquivo de entrada
        PrintWriter saída = null;  //declaração do arquivo de saída
        try {
            //Instancia leitor e gravador de arquivo
            entrada = new BufferedReader(new FileReader("c:/TEMP/arquivo.txt"));
            saída = new PrintWriter("c:/TEMP/arquivo_cópia.txt");
            //Rotina que lê caracter a caracter da entrada, gravando-os na saída
            String linha;
            while ((linha = entrada.readLine()) != null) {
                saída.println(linha);
            }
        } finally {
            //Fecha os arquivos, consolidando as leituras e gravações
            if (entrada != null) {entrada.close();}
            if (saída != null) {saída.close();}
        }
    }
}
```



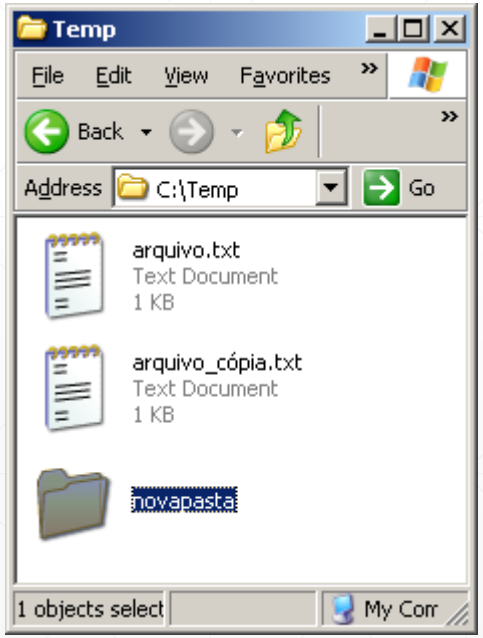
Criando Diretórios (Pastas)

Main.java

```
1   package jappio;
2
3   import java.io.File;
4
5   public class Main {
6       public static void main(String[] args) {
7           File pasta = new File("C:/TEMP/novapasta");
8           pasta.mkdir();
9       }
10  }
```

Saída - jappIO (run)

```
run:
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```



Listando arquivos de um diretório

Main.java

```
1  package jappio;
2
3  import java.io.File;
4
5  public class Main {
6      public static void main(String[] args) {
7          File pasta = new File("C:/TEMP");
8          if(pasta.isDirectory()){
9              String[] arquivos = pasta.list();
10             for(String arquivo: arquivos){
11                 System.out.println(arquivo);
12             }
13         }
14     }
15 }
```

Saída - jappIO (run)

```
run:
arquivo.txt
arquivo_cópia.txt
novapasta
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

Excluindo arquivo ou diretório

Main.java

```
1   package jappio;
2
3   import java.io.File;
4
5   public class Main {
6       public static void main(String[] args) {
7           File arquivo = new File("c:/TEMP/arquivo_cópia.txt");
8           if (arquivo.exists()){
9               boolean sucesso = arquivo.delete();
10              if(sucesso){
11                  System.out.println("Arquivo excluído!!!");
12              }
13          }
14      }
15  }
```

Saída - jappIO (run)

```
run:
Arquivo excluído!!!
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

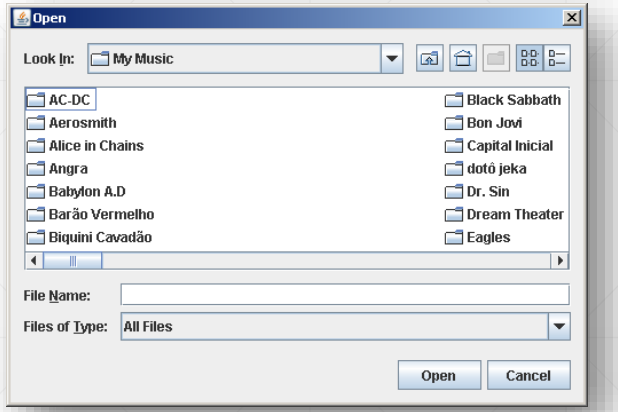
JFileChooser

- Classe que permite abrir uma caixa de dialogo onde o usuário pode indicar um diretório ou arquivo;
- Faz parte da biblioteca javax.swing, que é utilizada para desenvolvimento de aplicativos desktop.

JFileChooser

Main.java

```
1  package jappio;
2
3  import javax.swing.JFileChooser;
4  import java.io.File;
5
6  public class Main {
7      public static void main(String[] args) {
8          JFileChooser fc = new JFileChooser();
9          int retorno = fc.showOpenDialog(null);
10         if (retorno == JFileChooser.APPROVE_OPTION) {
11             File file = fc.getSelectedFile();
12             System.out.println("Abrindo: " + file.getName());
13         } else {
14             System.out.println("Cancelado pelo usuário");
15         }
16     }
17 }
```



JFileChooser – Selecionando um Diretório

```java
package jappio;

import javax.swing.JFileChooser;
import java.io.File;

public class Main {
    public static void main(String[] args) {
        JFileChooser fc = new JFileChooser();
        fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        int retorno = fc.showOpenDialog(null);
        if (retorno == JFileChooser.APPROVE_OPTION) {
            File pasta = fc.getSelectedFile();
            String arquivos[] = pasta.list();
            for(String arquivo: arquivos){
                System.out.println(arquivo);
            }
        } else {
            System.out.println("Cancelado pelo usuário");
        }
    }
}
```

```
Saída - jappIO (run)
run:
AC-DC
Aerosmith
AlbumArtSmall.jpg
```

Obrigado!