PROGRAMAÇÃO ORIENTADA A OBJETOS

PROF. EDSON MORENO

BASEADO NO MATERIAL GERADO PELO PROF. JULIO MACHADO

ARQUIVOS

ARQUIVOS

- Um **arquivo**
  - é um agrupamento de registros que seguem uma regra estrutural
  - Contém
    - informações (dados)
    - registrados em algum tipo de mídia
    - usualmente não volátil

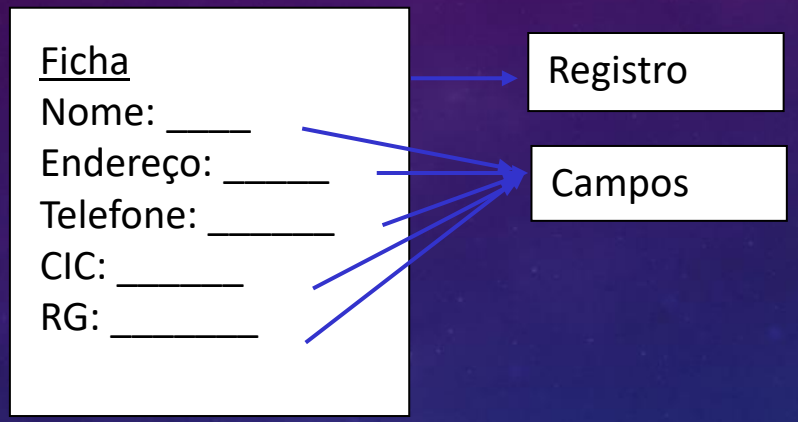
ARQUIVOS

- **Arquivos**
  - Podem conter informações de qualquer tipo de dados
    - textos, imagens, vídeos, programas, etc.
  - Organização e estrutura dos arquivos
    - Responsabilidade do Sistema Operacional
      - Organização em pastas
      - Visualização por parte do usuário

ARQUIVOS

- Arquivo (File)
  - Conjunto sequencial de registros relacionados
  - Exemplo: informações sobre um grupo de pessoas
- Registro (Record)
  - Conjunto de campos relacionados
  - Exemplo: nome, endereço, idade, telefone de uma pessoa
  - Pode ser representado por uma instância de uma classe
- Campo (Field)
  - Conjunto de caracteres com o mesmo significado
  - Exemplo: nome

ARQUIVOS



Campo

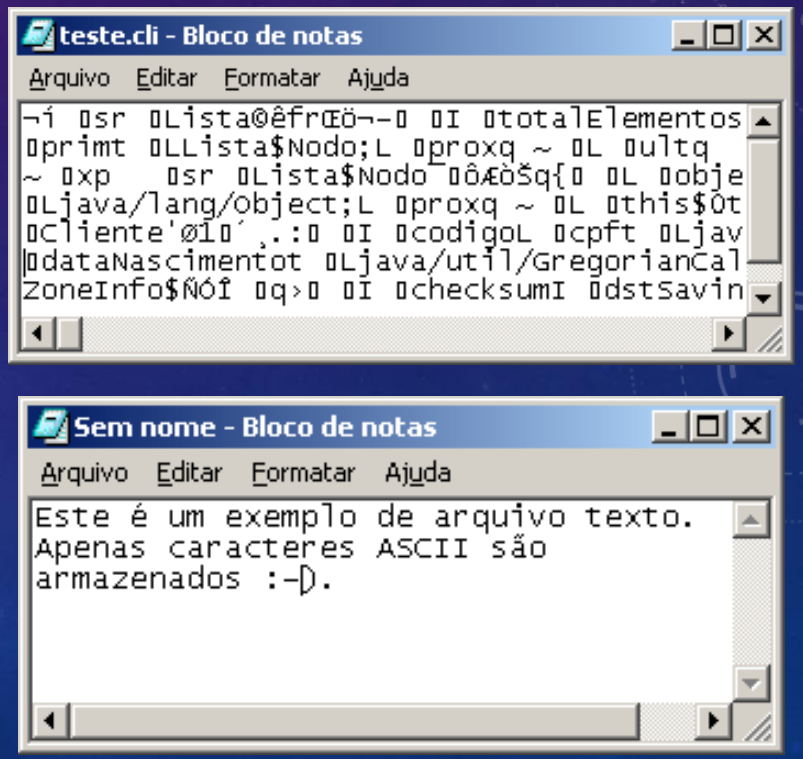
Registro

Nome	Endereço	Telefone	CIC	RG
Ana Silva	Andradas 34	332.56.35	345.565/9	271646252
João Neto	Siqueira 34	353.46.54	454.567/8	456546568
Maria Santos	Ipiranga 67	223.66.51	028.480/8	873260269
:	:	:	:	:

Arquivo

ARQUIVOS

- Arquivo binário
  - Os dados são armazenados em disco utilizando o mesmo formato binário de armazenamento em memória
- Arquivo texto
  - Os dados são armazenados em disco em registros de caracteres (usualmente UNICODE)



FLUXOS

SISTEMA DE ENTRADA E SAÍDA

- Criar um bom sistema de Entrada e Saída (E/S) é uma tarefa delicada na programação
  - Existem diversas abordagens
  - Devemos tratar várias origens e destinos para os dados (console, disco, impressora, conexão de rede,...)
  - Vários modos de acesso (sequencial, aleatório, com ou sem buffer, por linhas, por palavras, binário ou caractere,...)

API (*Application Programming Interface*)

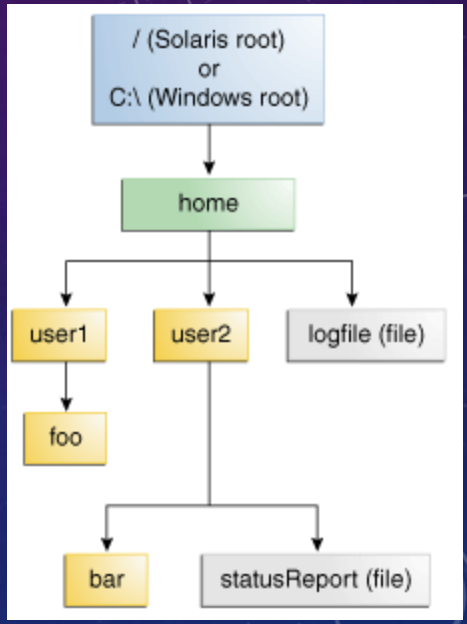
- Java provê uma biblioteca com muitas classes, cada uma com um propósito diferente
- Pacote `java.io.*`
  - Define E/S em termos de *streams* (fluxos)
  - *Streams* são sequências ordenadas de dados que possuem uma origem (streams de entrada) ou um destino (streams de saída)
- Pacote `java.nio.*`
  - Classes para a implementação do conceito de *Buffer*

API

- Diferentes classes para manipular um ***sistema de arquivos*** na API de Java:
  - *Path*
  - *File*
  - *Files*
  - etc

PATH

- A classe Path representa um caminho em um sistema de arquivos
  - Contém o nome de um arquivo e a lista de diretórios usada para construir o caminho para o arquivo
  - Exemplo: /home/user2/statusReport
  - É usada para examinar, localizar e manipular arquivos
  - Seu conteúdo depende do sistema operacional
    - Ex: no Windows, os caminhos podem começar pela letra de uma unidade (ex: "C:\")



PATH

•Criação de um caminho

```
Path path1 = Paths.get("/home/user2/statusReport");
Path path2 = Paths.get("/home","user2","statusReport");
Path path3 = Paths.get("c:\\temp\\dados.txt");
// Caminho a partir da raiz do usuário. Depende do S.O.
Path path4 = Paths.get(System.getProperty("user.home"),"books", "java7.pdf");
```

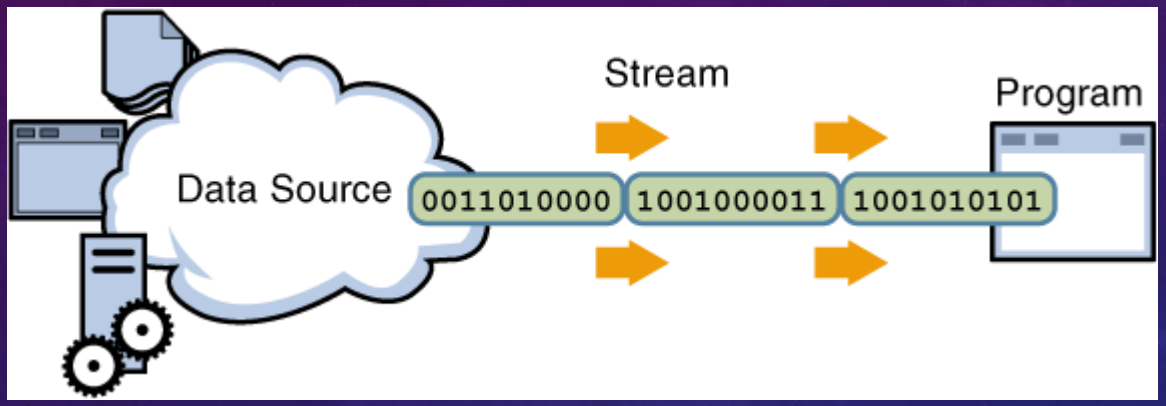
•Obtendo informações sobre o caminho:

```
System.out.println("toString: " + path1.toString()); // /home/user2/statusReport
System.out.println("getFileName: " + path1.getFileName()); // statusReport
System.out.println("getName(0): " + path1.getName(0)); // home
System.out.println("getNameCount: " + path1.getNameCount()); // 3
System.out.println("subpath(0,2): " + path1.subpath(0,2)); // /home/user2
System.out.println("getParent: " + path1.getParent()); // /home/user2
System.out.println("getRoot: " + path1.getRoot()); // / ou C:\
```

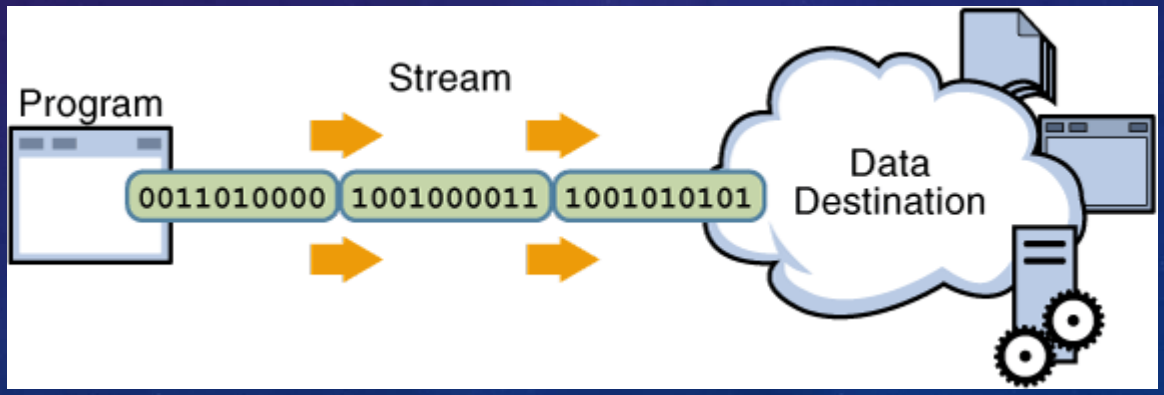
FILES

- A classe *Files* oferece ***métodos estáticos*** para a manipulação de arquivos:
  - Criar um novo arquivo
  - Ler dados do arquivo
  - Gravar dados no arquivo
  - Remover um arquivo
  - Alterar permissões

FLUXOS



Fluxo de leitura



Fluxo de escrita

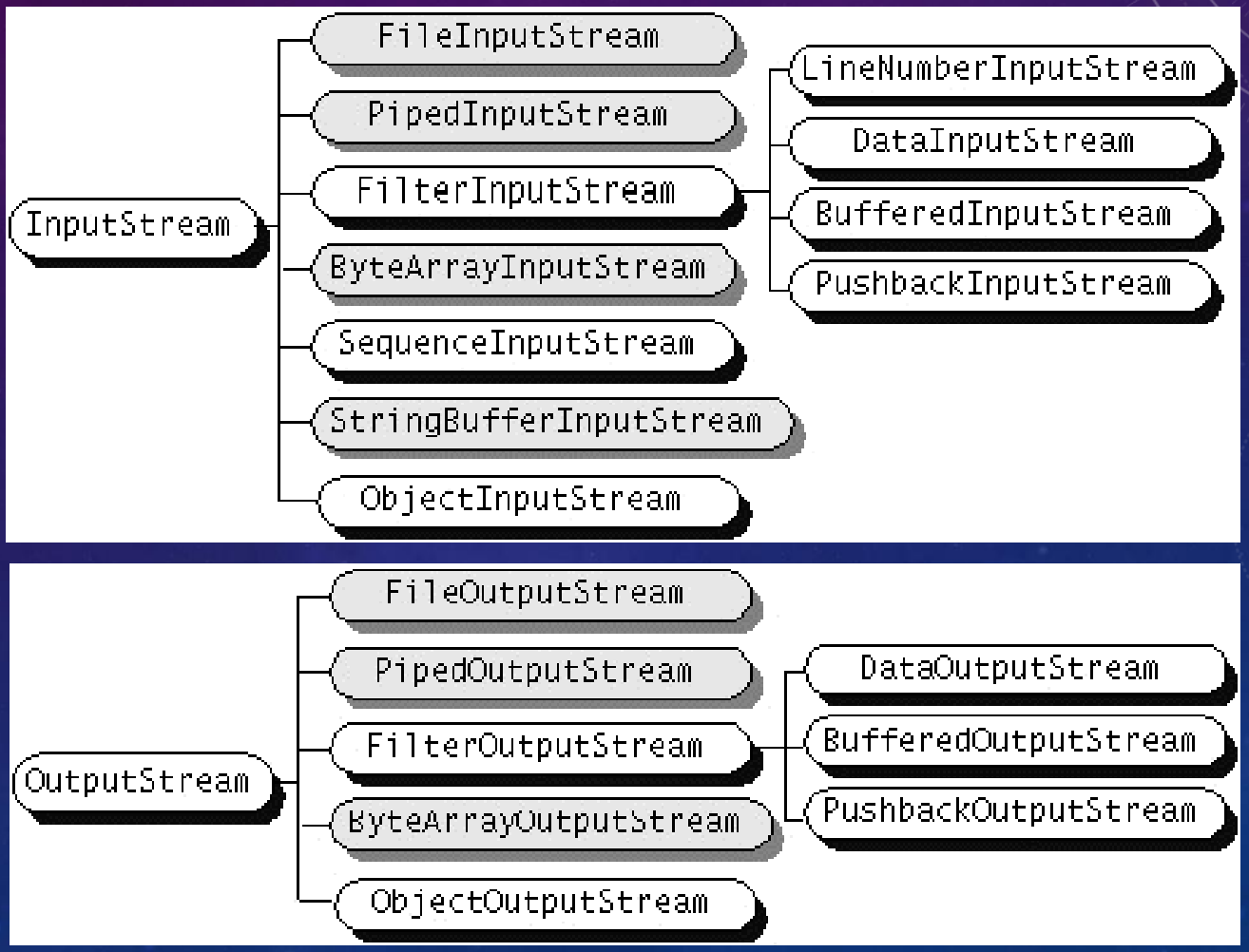
FLUXOS

- O pacote *java.io* fornece dois tipos de streams:
  - Fluxos de byte: tratam entrada ou saída de dados de 8 bits.
    - Para E/S baseada em dados binários
      - Ex.: uma imagem
    - Objetos `InputStream`, `OutputStream`
  - Fluxos de caractere: tratam entrada ou saída de caracteres Unicode.
    - Para E/S baseada em texto
      - Ex.: arquivo txt
    - Objetos `Reader`, `Writer`

FLUXOS

- Um *stream* é aberto quando é instanciado o objeto via construtor
- É possível ler e escrever do *stream* enquanto ele estiver aberto
  - Métodos podem gerar a exceção *IOException*
  - As exceções devem ser tratadas de algum modo pois são exceções verificadas
- Um *stream* é fechado quando se chama o método *close()*
  - É importante garantir que os fluxos sempre sejam fechados, ocorrendo ou não exceções

FLUXOS DE BYTES



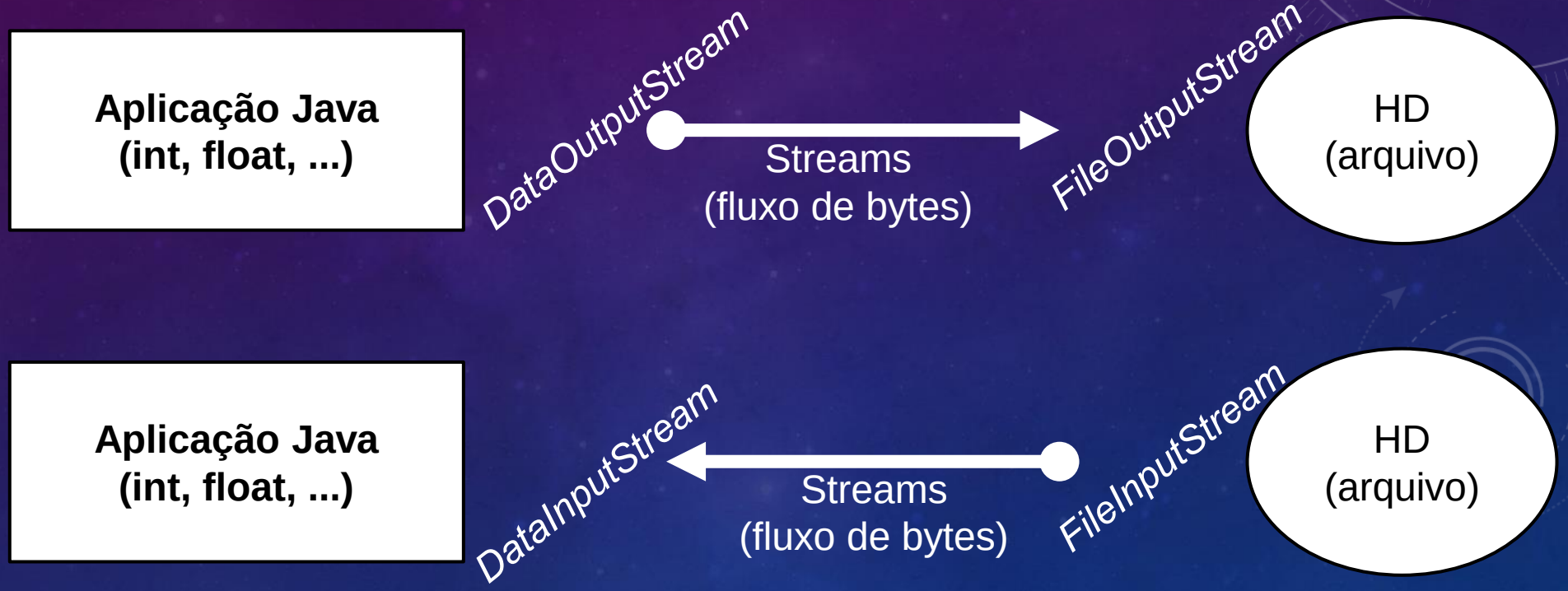
FLUXOS DE BYTES

- Fluxos de bytes:
  - São subclasses de `InputStream` e `OutputStream`
  - Possuem métodos:
    - `int read()` lê um só byte como um inteiro de 0 a 255 ou -1 se atingiu o final do fluxo
    - `void write(int b)` escreve um byte
  - Arquivos são abertos criando-se objetos das classes
    - `FileInputStream` (para leitura)
    - `FileOutputStream` (para escrita)

FLUXOS DE BYTES

- Leitura de dados diretamente como bytes:
  - Rápido, mas complicado
  - Usualmente lê-se dados como agregados de bytes que formam um *int*, um *double*, etc
  - Devemos utilizar classes de fluxos de dados conectadas no fluxo básico
    - Classes `DataInputStream` e `DataOutputStream`

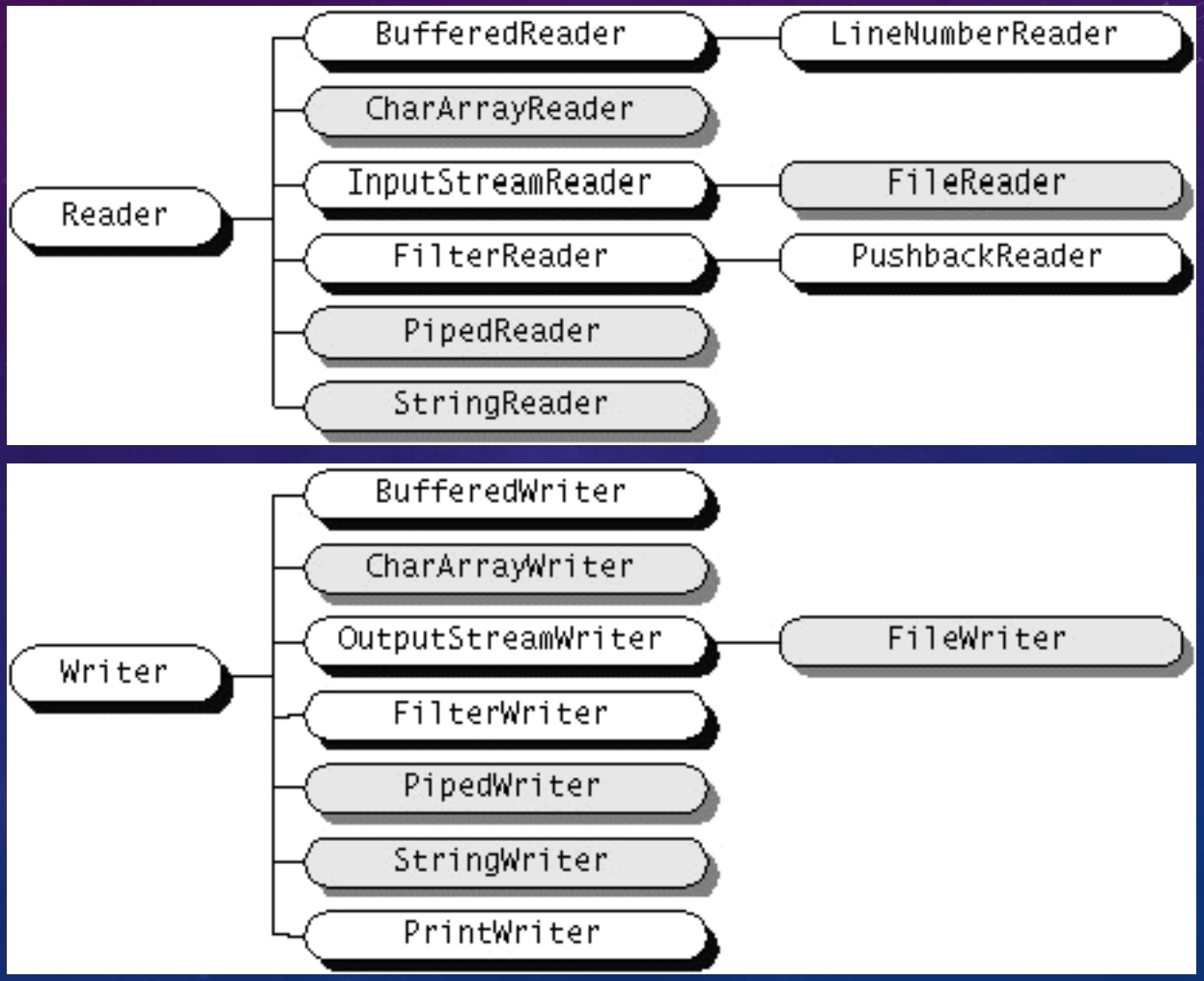
FLUXOS DE BYTES



FLUXOS DE BYTES

- As classes DataInputStream e DataOutputStream possuem métodos para E/S de inteiros, reais, etc
  - out.writeDouble(), out.writeChar(), out.writeLine(),...
  - in.readDouble(), in.readChar(), in.readLine(),...

FLUXOS DE CARACTERES



FLUXOS DE CARACTERES

- Fluxos de caracteres:
  - São subclasses de `Reader` e `Writer`
  - Possuem métodos:
    - `int read()` lê um único caractere como um inteiro ou -1 se atingiu o final do fluxo
    - `void write(int c)` escreve um caractere
  - Classes específica
    - `FileReader` acessam arquivos para leitura (read)
    - `FileWriter` acessam arquivos para escrita (write)

FLUXOS DE CARACTERES

- Podem ser associados filtros sobre os fluxos
  - Fornece meios de acesso mais sofisticadas
- Exemplos
  - A classe `BufferedReader` fornece um método para leitura de linhas de texto
  - A classe `PrintWriter` fornece métodos com formatação da saída de texto

CRIANDO UM ARQUIVO TEXTO

- Define-se **arquivo texto** como um arquivo onde a informação é organizada linha a linha
- A informação é gravada de forma textual (legível em qualquer editor)
- Duas opções:
  - **Forma livre**

```
Isto é um texto de forma livre
Onde cada linha tem um conteúdo e formato diferente
1, 2, 3, 4, 5, 6
```

  - **Formatado** (cada linha é um "registro")

```
Nome;DataNasc;CPF
Ana Freitas;08/12/1983;785639280.12
João da Silva;18/05/1998;847377330.47
Carlos Queiroz;23/10/2001;087628128.36
```

CRIANDO UM ARQUIVO TEXTO

```
public class DemoCriaArquivo{
  public static void main(String[] args){
    Random sorteia = new Random();

    String currDir = Paths.get("").toAbsolutePath().toString();
    String nameComplete = currDir+"\\"+"numeros.dat";
    Path path = Paths.get(nameComplete);

    try (PrintWriter writer =
          new PrintWriter(Files.newBufferedWriter(path, StandardCharsets.UTF_8))){
      for(int i=0;i<100;i++){
        int nro = sorteia.nextInt(1000);
        writer.print(nro+",");
      }
      int nro = sorteia.nextInt(1000);
      writer.print(nro+"\n");
    }catch (IOException x){
      System.err.format("Erro de E/S: %s%n", x);
    }
  }
}
```

CRIANDO UM ARQUIVO TEXTO

```
public class DemoCriaArquivo{
  public static void main(String[] args){
    Random sorteia = new Random();

    String currDir = Paths.get("").toAbsolutePath().toString();
    String nameComplete = currDir+"\\"+"numeros.dat";
    Path path = Paths.get(nameComplete);

    try (PrintWriter writer =
          new PrintWriter(Files.newBufferedWriter(path, StandardCharsets.UTF_8))){
      for(int i=0;i<100;i++){
        int nro = sorteia.nextInt(1000);
        writer.print(nro+",");
      }
      int nro = sorteia.nextInt(1000);
      writer.print(nro+"\n");
    }catch (IOException x){
      System.err.format("Erro de E/S: %s%n", x);
    }
  }
}
```

Obtém uma instância da classe “Path”. A classe “Path” armazena a localização do arquivo na unidade de armazenamento.

CRIANDO UM ARQUIVO TEXTO

```
public class DemoCriaArquivo{
  public static void main(String[] args){
    Random sorteia = new Random();

    String currDir = Paths.get("").toAbsolutePath().toString();
    String nameComplete = currDir+"\\"+"numeros.dat";
    Path path = Paths.get(nameComplete);

    try (PrintWriter writer =
         new PrintWriter(Files.newBufferedWriter(path, StandardCharsets.UTF_8))){
      for(int i=0;i<100;i++){
        int nro = sorteia.nextInt(1000);
        writer.print(nro+",");
      }
      int nro = sorteia.nextInt(1000);
      writer.print(nro+"\n");
    }catch (IOException x){
      System.err.format("Erro de E/S: %s%n", x);
    }
  }
}
```

Cria a entrada do arquivo no caminho indicado por "path". Se já existe arquivo com o mesmo nome ele será sobrescrito

CRIANDO UM ARQUIVO TEXTO

```
public class DemoCriaArquivo{
  public static void main(String[] args){
    Random sorteia = new Random();

    String currDir = Paths.get("").toAbsolutePath().toString();
    String nameComplete = currDir+"\\"+"numeros.dat";
    Path path = Paths.get(nameComplete);

    try (PrintWriter writer =
         new PrintWriter(Files.newBufferedWriter(path, StandardCharsets.UTF_8))){
      for(int i=0;i<100;i++){
        int nro = sorteia.nextInt(1000);
        writer.print(nro+",");
      }
      int nro = sorteia.nextInt(1000);
      writer.print(nro+"\n");
    }catch (IOException x){
      System.err.format("Erro de E/S: %s%n", x);
    }
  }
}
```

Grava uma string no arquivo. Segue as mesmas regras do “print/println”.

CRIANDO UM ARQUIVO TEXTO

- Quando o programa deixa o “bloco try” o arquivo é fechado automaticamente e os dados estão prontos para serem acessados por qualquer programa;
- Note que os números foram “gravados” em uma única linha separados por “,”; Analise o conteúdo do arquivo usando:
  - Um programa editor de texto (notepad, atom, VSCode, Vi, Vim etc)
  - Os comandos do shell: “type” (DOS) ou “cat” (Linux)

LENDO UM ARQUIVO TEXTO

```
public class DemoLeArquivo {

  public static void main(String args[]){

    String currDir = Paths.get("").toAbsolutePath().toString();

    String nameComplete = currDir+"\\"+"numeros.dat";

    Path path = Paths.get(nameComplete);


    int acum = 0;

    double media = 0.0;

    String linha = "";

    try (Scanner sc =

            new Scanner(Files.newBufferedReader(path, StandardCharsets.UTF_8))){

      linha = sc.nextLine();

    }catch (IOException x){

         System.err.format("Erro de E/S: %s%n", x);

    }

    String[] numeros = linha.split(",");

    for(int i=0;i<numeros.length;i++){

      acum += Integer.parseInt(numeros[i]);

    }

    media = acum/numeros.length;

    System.out.println("Quantidade de valores lidos: "+numeros.length);

    System.out.println("Somatorio: "+acum);

    System.out.println("Media: "+media);

  }

}
```

Obtém caminho para o arquivo

Lê todos os números para uma única string

Usa o método “split” da classe String para gerar um array com todos os “tokens” separados por “,”

LENDO UM ARQUIVO TEXTO

- A classe Scanner pode ser usada para ler dados do teclado ou de um arquivo texto. No caso do teclado o “arquivo” se chama “System.in”;
- Como os números foram armazenados em uma única linha, separados por vírgula, a leitura foi feita por um único comando “nextLine”;

TRABALHANDO COM REGISTROS

- O exemplo disponível no moodle junto a estes slides demonstra como trabalhar com registros;
- Antes de executar certifique-se que o arquivo “produtos.txt” encontra-se na mesma pasta que o programa (consulte o conteúdo do mesmo com um editor de textos; acrescente ou remova produtos se quiser);
- Para trabalhar com “registros” identificamos o final de cada linha com um “\n”. Dessa forma podemos ler uma linha de cada vez;
- Cada linha é recuperada como uma string. Para separar os dados usamos o método “split”.

TRABALHANDO COM REGISTROS

- Cada vez que se lê uma linha, o “cursor” do arquivo avança automaticamente. Dessa forma fazemos uma leitura sequencial do arquivo sem precisarmos saber em que linha estamos ou quantas linhas o arquivo tem.
- Arquivos texto foram feitos para serem carregados na memória e depois regravados. As alterações dos dados devem ser feitas nas estruturas na memória e depois o arquivo é sobrescrito com os dados alterados.
- Quando é necessário alterar dados diretamente no arquivo a melhor opção é usar arquivos binários ou um sistema gerenciador de banco de dados.