Disciplina: LP II
Prof. Lucas Gonçalves Nadalete

Aula 12 – Manipulação de Arquivos

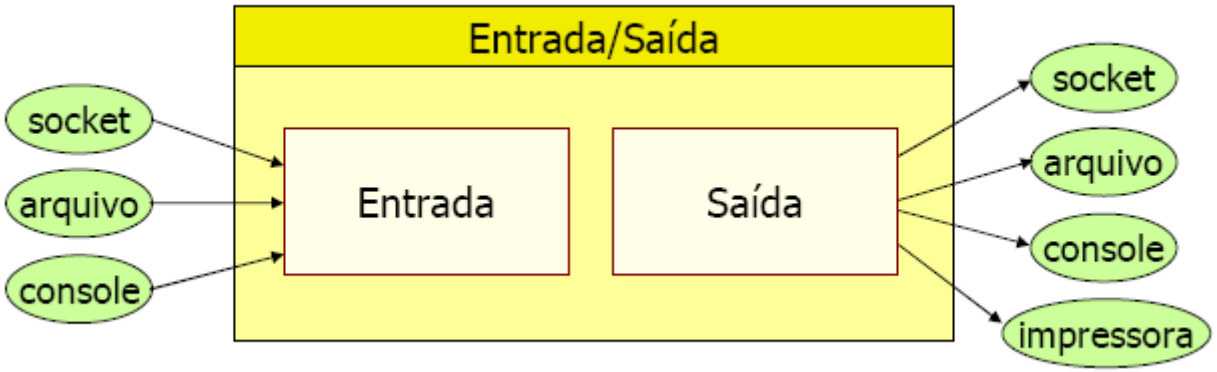
Introdução

- Java tem uma API específica para operações de I/O (Entrada e Saída) de arquivos
- As operações de INPUT e OUTPUT facilitam a interação dos programas Java com o mundo externo:
  - podem ser externo como teclado e mouse, ou interno, tal como a placa de rede e disco rígido.

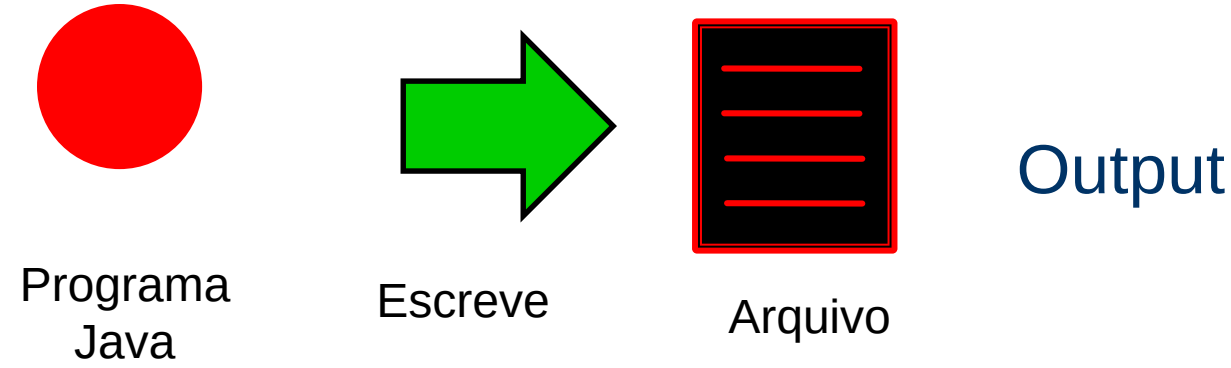
Introdução

- Qualquer aplicação computacional pode realizar manipulações de dados
- Dados podem ser adquiridos/salvos pela aplicação de diferentes formas
  - Terminal, arquivos, impressora, etc
- Em Java a entrada/saída é processada por classes do pacote java.io que deve ser importado

Entrada e Saída (Input e Output)



Entrada e Saída



Fluxo Entrada/Saída

- A entrada e saída de dados em Java é realizada através de streams
- Uma stream é uma sequência ordenada de bytes ou caracteres de tamanho indefinido
- Streams podem ser abertas e fechadas pelo programa; algumas vezes, estas operações são feitas de forma automática (geralmente em caso de erro).

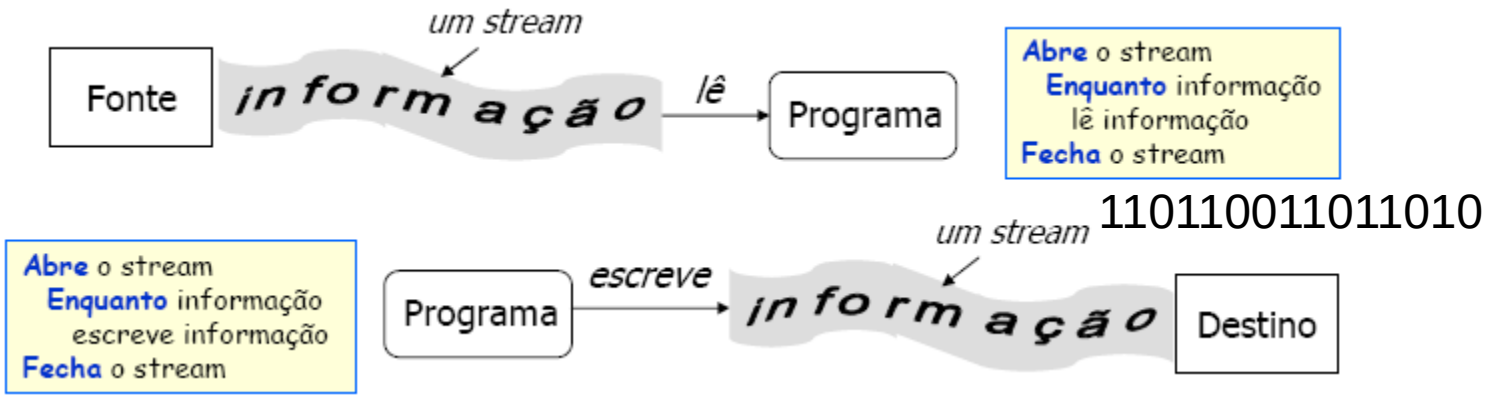
Fluxo Entrada/Saída

- Streams lidam apenas com entrada ou saída de bytes.
- Para podermos tratar com tipos mais complexos precisamos associar filtros a um stream
- InputStream (Entrada)
- OutputStram (Saída)
  - Transformam bytes em chars

Fluxo Entrada/Saída

teclado, arquivo,
socket..

110110011011010



arquivo, socket,
console, impressora...

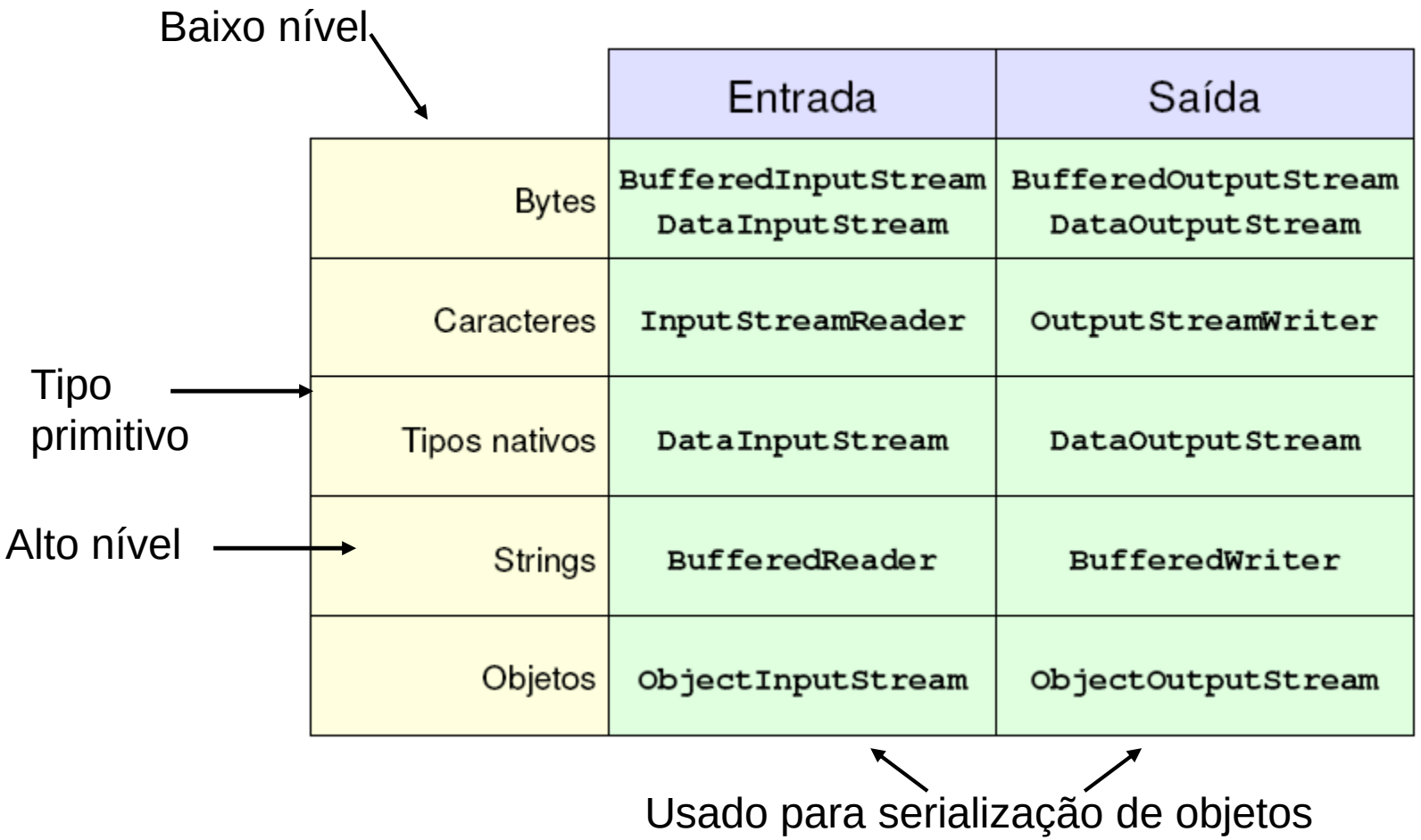
Tipos de Fluxos

- O fluxo de dados entre a aplicação e o arquivo pode ser de entrada ou de saída
- Fluxo de Entrada: a aplicação lê dados do arquivo e armazena em uma variável
- Fluxo de Saída: a aplicação escreve o valor de uma variável no arquivo

Entrada e Saída - Arquivos

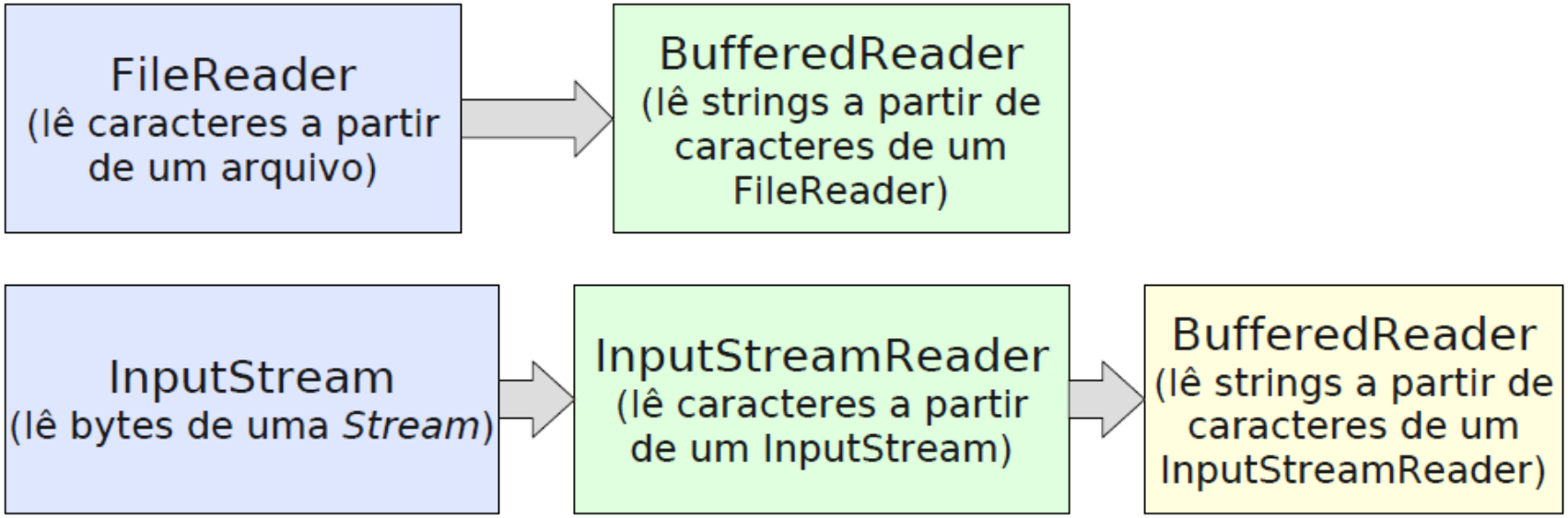
- Há diferentes formas para ler e escrever dados:
  - sequencialmente, aleatoriamente, como bytes, como caracteres, linha por linha, palavra por palavra...
  - Em Java, basicamente, existem dois grupos:
    - Entrada e Saída de bytes:
      - InputStream e OutputStream;
    - Entrada e Saída de caracteres (chars):
      - Reader e Writer.

Classes para Entrada e Saída



	Entrada	Saída
Bytes	`BufferedInputStream` `DataInputStream`	`BufferedOutputStream` `DataOutputStream`
Caracteres	`InputStreamReader`	`OutputStreamWriter`
Tipos nativos	`DataInputStream`	`DataOutputStream`
Strings	`BufferedReader`	`BufferedWriter`
Objetos	`ObjectInputStream`	`ObjectOutputStream`

IO em Java: Classes



As classes mais utilizadas para leitura e escrita são: BufferedReader e PrintWriter que permitem a manipulação dos dados em alto nível de abstração por meio da leitura e escrita de Strings

Arquivos

- Outro dispositivo de entrada e saída de vital importância é manipulado através do conceito de arquivo
- Um arquivo é uma abstração utilizada para uniformizar a interação entre o ambiente de execução e os dispositivos externos

Arquivos

- A interação de um programa com um dispositivo através de arquivos passa por três etapas:
  - abertura ou criação de um arquivo
  - transferência de dados
  - fechamento do arquivo

A Classe File

- Em Java a classe File permite representar arquivos neste nível de abstração
  - File objetoFile = new File (“nomeArquivo”);
- A maioria dos métodos desta classe lança a exceção da super classe IOException
  - Outros: FileNotFoundException, DirectoryNotFoundException, PathTooLongException ...

Classe 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 diversos métodos para testar a existência de arquivos, apagar arquivos, criar diretórios, listar o conteúdo de diretórios, etc..

Alguns métodos Classe File

- String getParent(); retorna o diretório (objeto File) pai
- list(); retorna lista de arquivos contidos no diretório
- boolean isFile(); retorna se é um arquivo
- boolean isDirectory(); retorna se é um 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 getPath(); retorna o caminho a partir da raiz
- String getAbsolutePath(); Ex. C:/Localizacao/../file.txt
- String getName();

Exemplos da Classe File

- File diretorio = new File("c:\\novoDiretorio");
- diretorio.mkdir(); // cria o dir, se possível
- File subdir1 = new File(diretorio,“subdir1");
- subdir1. mkdir();
- File subdir2 = new File(diretorio,"subdir2");
- subdir2. mkdir();
- File arquivo = new File(diretorio,
  “NovoArquivo.txt");
- FileWriter f = new FileWriter(arquivo);
- f.close();

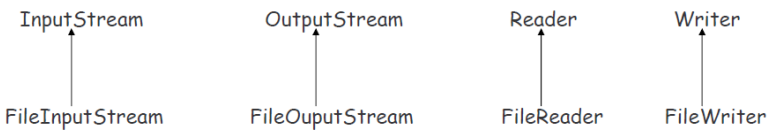
Limitação da Classe File

- Porém, a classe File não suporta manipulação de entrada e saída de dados
- Existem outras classes Específicas para isto:
  - Ex. FileReader, FileWriter, FileInputStream e FileOutputStream

Classes da API/IO

- Os arquivos são abertos criando-se objetos destas classes de fluxo que herdam de InputStream, OutputStream, Reader, Writer.

Classe Abstrata



Classe Concreta

Principais Classes da API/IO

FileInputStream	para entrada baseada em bytes de um arquivo
FileOutputStream	para saída baseada em bytes de um arquivo
FileReader	para entrada baseada em caracteres de um arquivo
FileWriter	para saída baseada em caracteres de um arquivo

As classes acima oferecem pelo menos um construtor que recebe como argumento um objeto da classe File e implementam os métodos básicos de transferência de dados.

Lendo do Teclado

```
public static void main (String args[])
    throws FileNotFoundException, IOException {
    System.out.print("Digite o texto");
    FileOutputStream fO = new FileOutputStream("C:\\TesteSaida.txt");
    byte a = (byte) System.in.read();
    while (a != '\n') {
        fO.write(a);
        a = (byte) System.in.read();
    }
}
```

Escrevendo com tipos primitivos

```
public static void main (String args []) throws IOException {

 try {
        DataOutputStream w =
                new DataOutputStream(new FileOutputStream("NomeArquivo.dat"));
        w.writeInt(74); // 4 bytes
        w.writeChar('a'); // 2 bytes
        w.writeBoolean(true); // 1 byte
        w.close();
        System.out.println(w.size());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Nível intermediário,
manipulando tipos primitivos

1 byte: **boolean**, **byte**
2 bytes: **char**, **short**
4 bytes: **int**, **float**
8 bytes: **long**, **double**

Leitura Sequencial - FileWriter

- Uma maneira bastante eficiente de ler um arquivo de texto é usar FileReader com um BufferedReader

Alto nível, estamos manipulando Strings

```
public static void main (String args []) throws IOException {

   FileReader f = new FileReader("c:\\arq.txt");
    BufferedReader in = new BufferedReader(f);
    String linha = in.readLine();
    while (linha != null) {
        System.out.println(linha);
        linha = in.readLine();
    }
    in.close();
}
```

Lê linha do arquivo

Ponteiro aponta para próxima linha

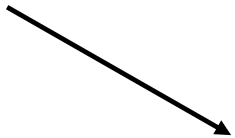
Libera recursos

Exemplo: Gravação Sequencial - FileWriter

```
public static void main (String args[]) throws FileNotFoundException, IOException {

    File arquivo = new File("C:\\TesteSaida.txt");
    FileWriter wr = new FileWriter(arquivo);
    PrintWriter out= new PrintWriter(wr);
    out.println("linha1");
    out.println("linha2");
    out.close();


}
```



Se não fechar o arquivo, não terá o conteúdo linha1 e linha 2, o arquivo ficará em branco

Exercício

- Dado dois arquivos arq1 e arq2, desenvolva um programa em Java que implemente as seguintes funcionalidades:
  - cópia linha a linha do conteúdo do arq1 para arq2 (alto nível);
  - imprime na tela todo o conteúdo do arquivo arq2;
- Obs.: os arquivos arq1 e arq2 são arquivos do tipo texto

Class RandomAcessFile

- Construindo uma instância do RandomAccessFile, você pode procurar por qualquer posição desejada dentro de um arquivo, e então ler ou escrever um montante de dados desejados.
- Esta classe oferece acesso aleatório através do uso de um ponteiro.
- Na prática, esta classe é pouco utilizada visto que, em geral, o acesso aos arquivos necessitam ser sequencias

Class RandomAcessFile

- Construindo uma instância do RandomAccessFile no modo 'r', se o arquivo não existir dispara uma exceção "FileNotFoundException".
- Construindo uma instância do RandomAccessFile no modo 'rw', se o arquivo não existir um arquivo de tamanho zero é criado.

Class RandomAcessFile

```
RandomAccessFile arquivo =
        new RandomAccessFile(new File("c:\\ arquivo.txt"),"rw");
RandomAccessFile arquivo2 =
         new RandomAccessFile(new File("c:\\ arquivo.txt"),"r");
   arquivo.writeInt(1); // 4 byte
   arquivo.writeDouble(1984); // 8 bytes
   arquivo.writeChar('T'); // 2 // bytes
  //Leitura
  System.out.println(arquivo2.readInt());
  System.out.println(arquivo2.readDouble());
  System.out.println(arquivo2.readChar());

 //Volta com o ponteiro de leitura ao inicio do arquivo para fazer a leitura do int
  arquivo2.seek(0);
  System.out.println(arquivo2.readInt());

//Joga o ponteiro para a leitura do char, na posição 12 (4 do int e 8 do double)
   arquivo2.seek(12);
   System.out.println(arquivo2.readChar());
```

Serialização de Objetos

- Java permite a gravação direta de objetos em disco ou seu envio através da rede
- De forma genérica a serialização é uma técnica usada para persistir objetos
- Com isso, pode-se gravar objetos em disco, fazer a transmissão remota de objetos via rede, armazenar os objetos em arquivos

Serialização de Objetos

- A serialização é o processo de armazenar um objeto, incluindo todos os atributos públicos e privados para um stream (transformar em binário) dentro de um arquivo
- O estado de um objeto é salvo, de modo que ele possa ser recuperado numa ocasião futura

Serialização

- Armazenamento de instâncias de classes: persistência.
- Mecanismo "manual":
  - Métodos que armazenam e recuperamos valores de todos os campos.
- Mecanismo automático: serialização.
  - Basta declarar que a classe a ser armazenada implementa a interface Serializable.
  - Não precisar implementar nenhum método, é apenas uma marcação que a classe é mesmo serializável

Serialização

- Se uma classe pai é Serializada, então qualquer subclasse desta também será serializada por default
- Algumas classes não são serializáveis
  - Exemplos: Socket, ResultSet...
- Ao tentar gravar um objeto cuja classe não é serializável ocasionará um erro de compilação!

Salvando objetos

```
public static void main (String args []) throws IOException {

  Pessoa a = new Pessoa("Cividanes");
  Pessoa b = new Pessoa("Alonso");
  File arquivo = new  File("c:\\SalvandoObjetos.txt");
  FileOutputStream fOut = new FileOutputStream(arquivo);
  ObjectOutputStream objOut = new ObjectOutputStream(fOut);
  objOut.writeObject(a);
  objOut.writeObject(b);

}
```

Gravação dos objetos a e b no arquivo

Esta classe deve implementar **Serializable** ou ser subclasse de uma que faça tal implementação

Lendo os Objetos Salvos

- Como ler o objeto salvo no arquivo anterior?
- Dica: Utilize os objetos FileInputStream,ObjectInputStream e o método readObject()

Lendo Objeto Salvo

```
//Carrega o arquivo
 FileInputStream arquivoLeitura
         = new FileInputStream("c:\\saida.dat");
 //Classe responsavel por recuperar os objetos do arquivo
 ObjectInputStream objLeitura =
                              new ObjectInputStream(arquivoLeitura);
 Cliente cliente = (Cliente) objLeitura.readObject();
 System.out.println("CPF: "+ cliente.getCpf());
 System.out.println("Nome: " + cliente.getNome());
 objLeitura.close();
 arquivoLeitura.close();
```

Exercício

- Dados dois arquivos x1 e x2 contendo números já ordenados
- Criar um arquivo x3 contendo uma união dos números contidos em x1 e x2 também ordenados, porém sem números repetidos
- Utilize a API alto nível do para nagevação de I/O em Java: FileReader e BufferedReader, FileWriter e PrintWriter