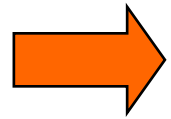
IO e Serialização

Parte I

O pacote java.io

- Em Java podemos encontrar recursos para facilitar:
  - A manipulação de dados durante o processo de leitura ou gravação
  - Bytes sem tratamento
  - Caracteres Unicode
  - Dados filtrados de acordo com certo critério
  - Dados otimizados em buffers
  - Leitura/gravação automática de objetos
  - Entre outras coisitas a mais...



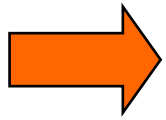
I/O em Java

- Em Java, as informações são:
  - Armazenada ou gravadas e apanhada ou lidas usando um sistema de comunicação chamado *streams*.

- É possível criar:
  - *streams* de entrada para ler informações e
  - *streams* de saída para armazenar informações.

- ....ou seja, os *streams* trabalham com o tráfego da informação
  - Seja a informação de disco, da Internet, teclado ou de outros programas.

Mas que raios são *streams*?.....

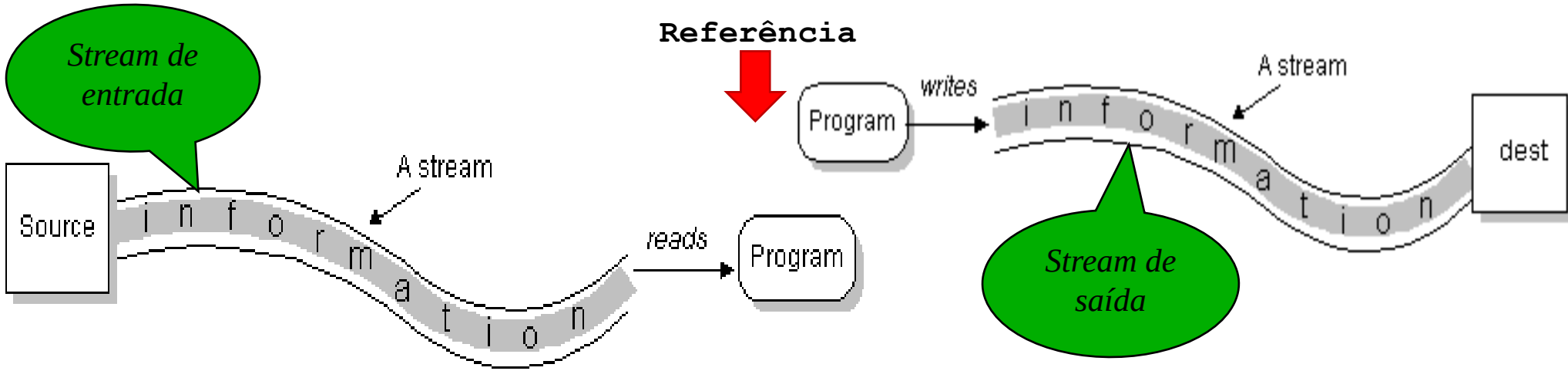


I/O em Java

- Mas, o que são *streams*?
  - Um *stream* é o caminho atravessado pelos dados em um programa

⇨ Um *stream* de entrada
  - Envia dados de uma origem para um programa

⇨ Um *stream* de saída
  - Envia dados de um programa para um destino.



I/O em Java

- Os *Streams* que podemos trabalhar em Java são:

Streams de *bytes*

- São usados para lidar com *bytes*, inteiros e outros tipos de dados simples
- Ex:. Programas executáveis, comunicações pela Internet e *bytecode* utilizam esse *stream*

Streams de caracteres

- Tratam de arquivos textos e outras fontes de textos
- Ex:. São um tipo especializado de *stream* de *bytes*, que trata somente de dados textuais, tais como arquivos texto, páginas Web como documentos HTML, dados do usuário, etc.

I/O em Java

- Os Streams que vamos trabalhar em Java são:

Streams de dados

- Permitem escrita e leitura de tipos primitivos diretamente (char, float, integer, double, etc)

- Ex:. Se precisar trabalhar com dados que não sejam representados como bytes ou caracteres, podemos usar os *streams* de entrada e saída de dados.
  - **Estes *streams* filtram um *stream* de bytes existente de modo que os tipos primitivos possam ser lidos ou escritos diretamente do *stream***

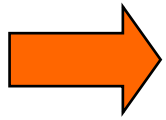
Streams de objetos

- Permite que os dados sejam representados como parte de um objeto

- Ex:. Tratam da persistência e recuperação de objetos como um todo para que um objeto seja salvo em um destino, como um arquivo de disco, por exemplo.

I/O em Java

- Blz,
  - Entendemos o que é um stream e os tipos que temos…
  - Mas como podemos efetivamente utilizar esses *stream*…???



I/O em Java

- Usando um *stream*
  - O procedimento para usar um *stream* de *byte* ou caracteres em Java é praticamente o mesmo
    - Vamos entender o processo de criação e uso de *streams*.

⇨ Para um *stream* de entrada.

- O primeiro passo é criar um objeto que esteja associado à origem de dados.
  - Ex:. Se a origem for um arquivo no disco, um objeto *FileInputStream* poderia ser associado a esse arquivo

- Depois que o objeto *stream* estiver associado, é possível ler informações a partir desse *stream*, usando um dos métodos do objeto
  - Ex:. No caso do exemplo acima, o método *read().*

I/O em Java

- Usando um *stream*

➡ Para um *stream* de saída.

- O primeiro passo é criar um objeto que esteja associado ao destino dos dados.
  - Ex:. Um objeto desse tipo pode ser o *BufferedWritter*

- Depois que o objeto *stream* estiver associado, é possível escrever informações a partir desse *stream*, usando um dos métodos do objeto
  - Ex:. Método *writer()*

➡ Quando terminar de ler ou escrever os dados,

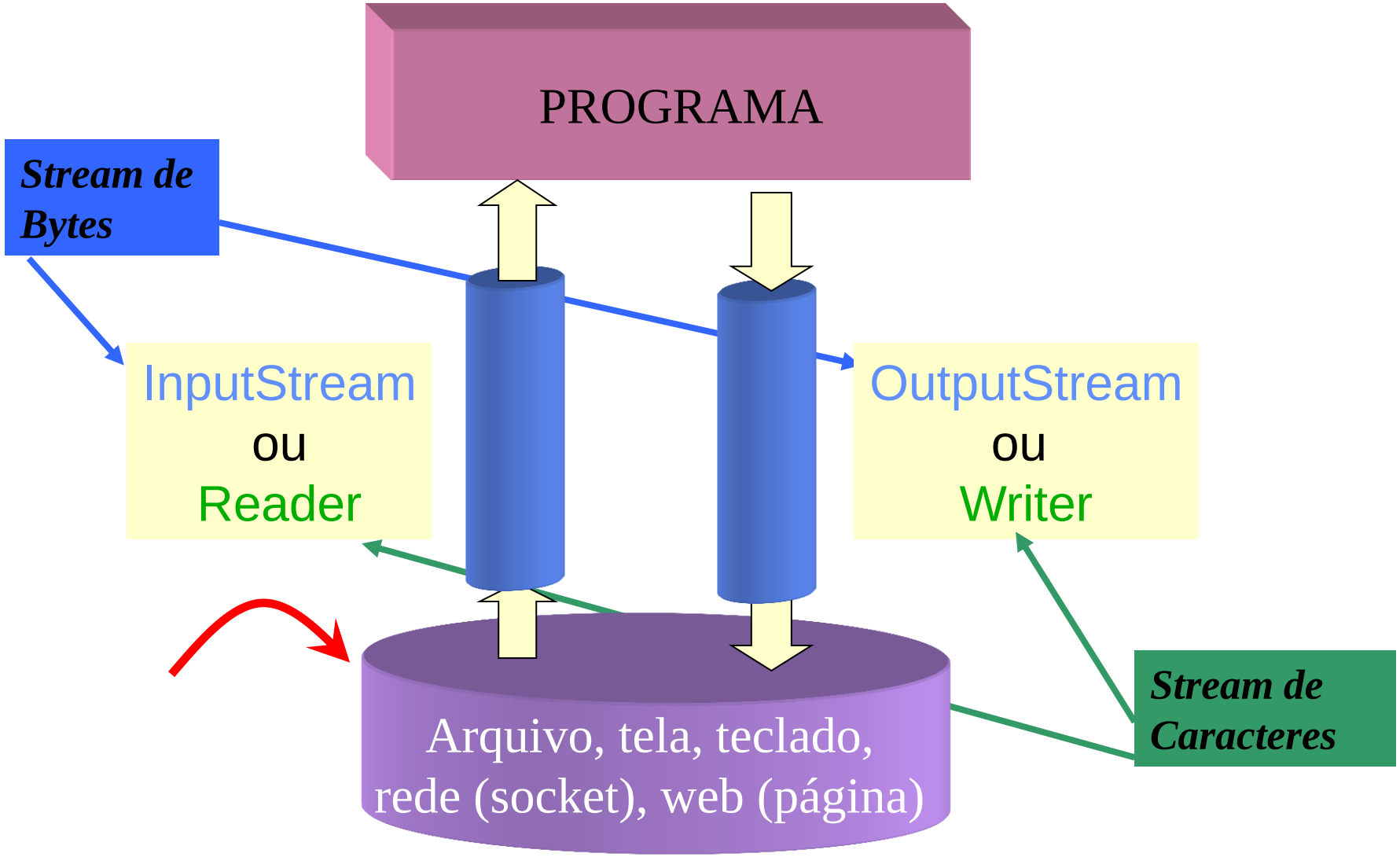
- Basta chamar o método *close()*, para indicar que terminou de usar o *stream*.

I/O em Java

- Usando um *stream*
  - **Resumindo:** Para usar um *stream* basta:
    - ⇨ Criá-lo e
    - ⇨ Chamar os seus métodos para enviar ou receber dados,
      - Dependendo se é um *stream* de entrada ou saída.

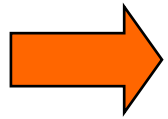
I/O em Java

- Resumindo: Fluxo dos dados e as classes utilizadas



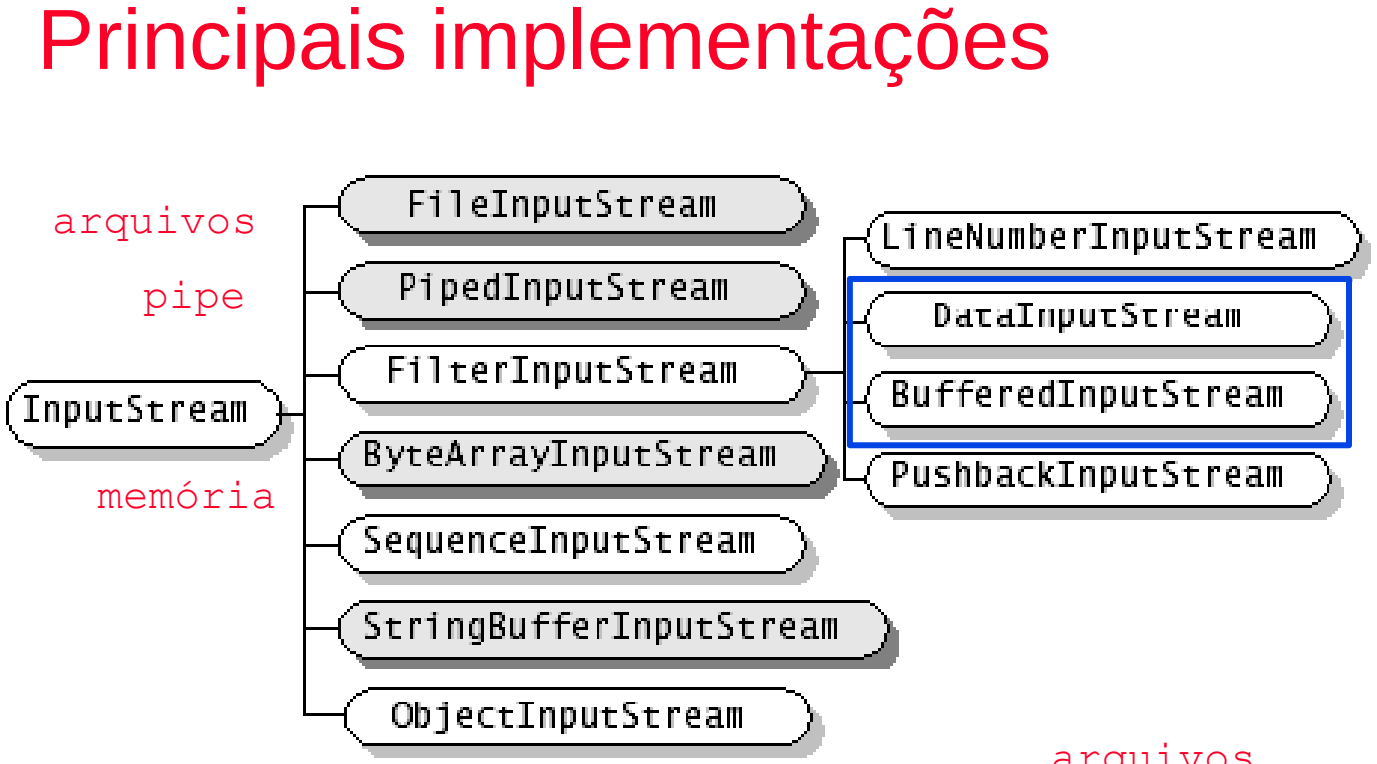
Hierarquia de Classes

- Blz,
  - Mas quais são as classes concretas com as quais a gente vai ter que trabalhar???...
  - Para responder essa questão temos que analisar rapidamente a hierarquia de classes dessa api.....



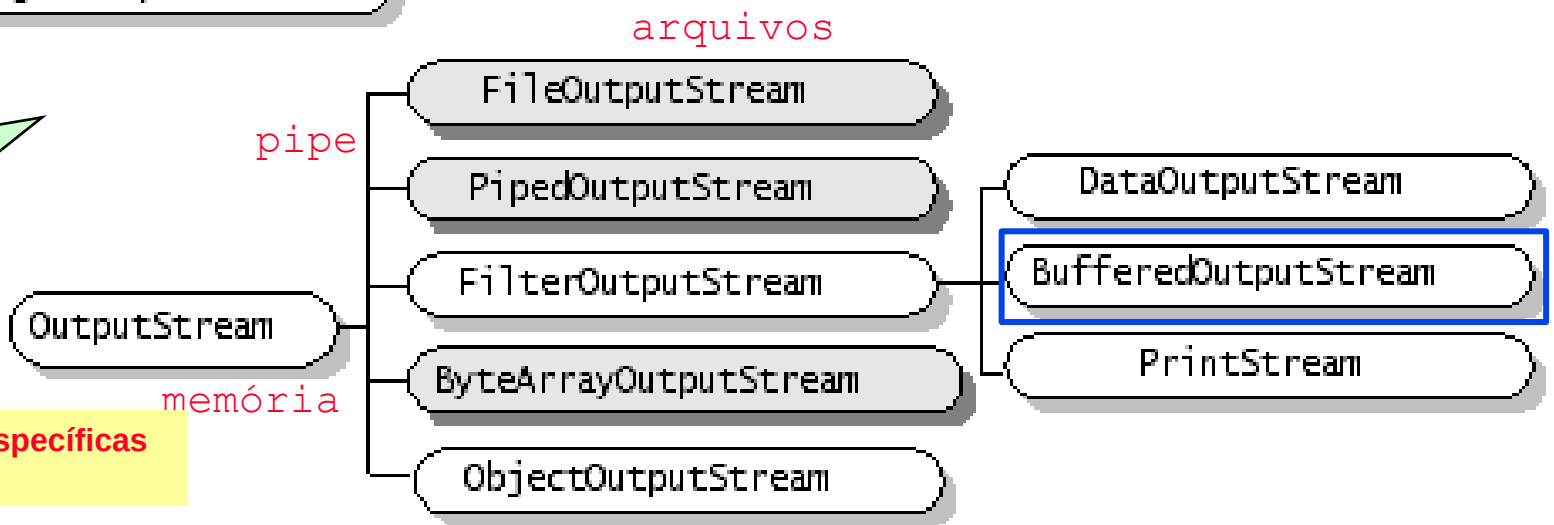
Hierarquia: InputStream, OutputStream

- Principais implementações



As classes iniciais são abstratas, logo é necessário definir as subclasses que deverão ser instanciadas, montando essa hierarquia.

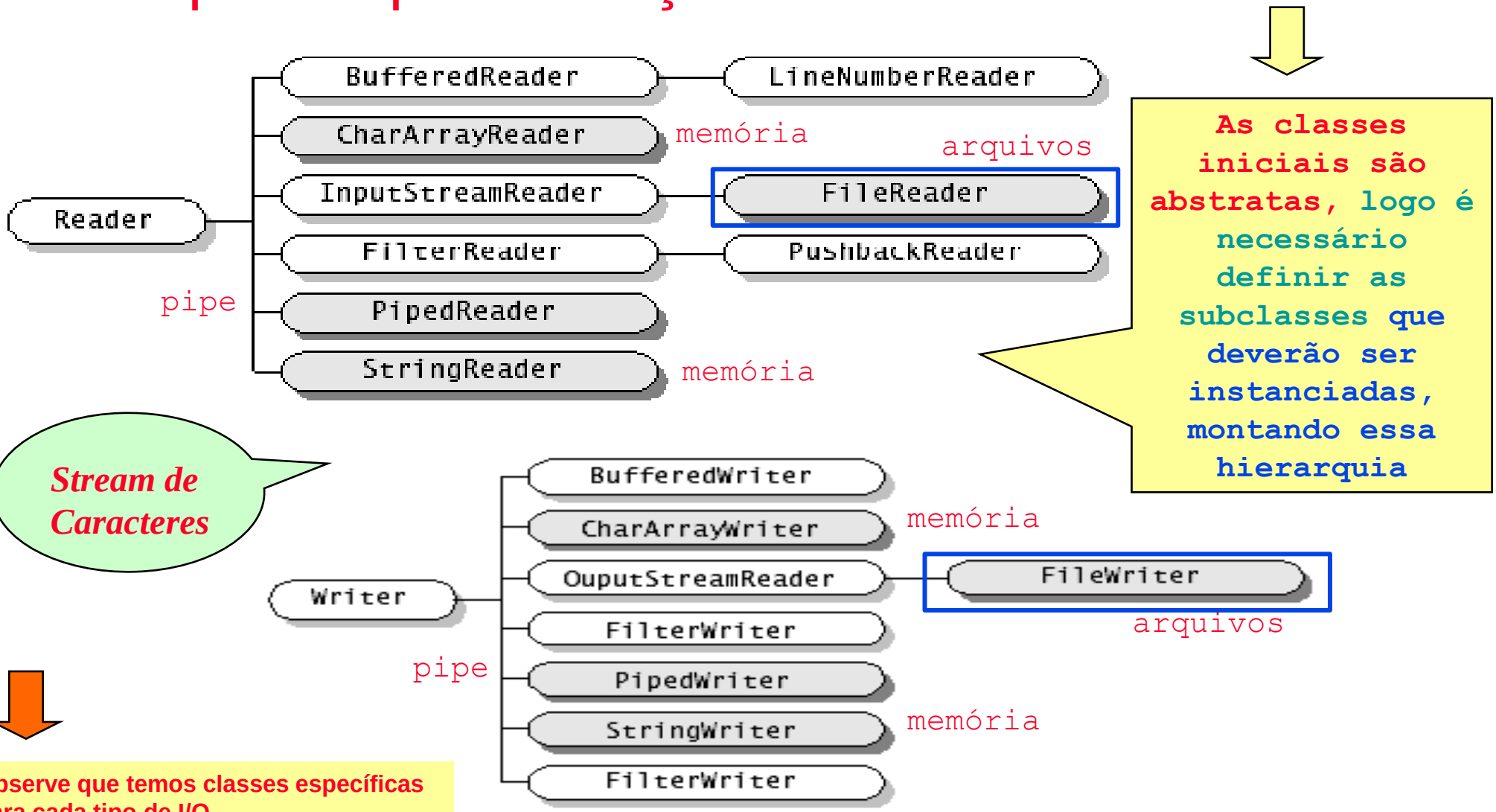
Stream de Bytes



Observe que temos classes específicas para cada tipo de I/O

Hierarquia: Reader, Writer

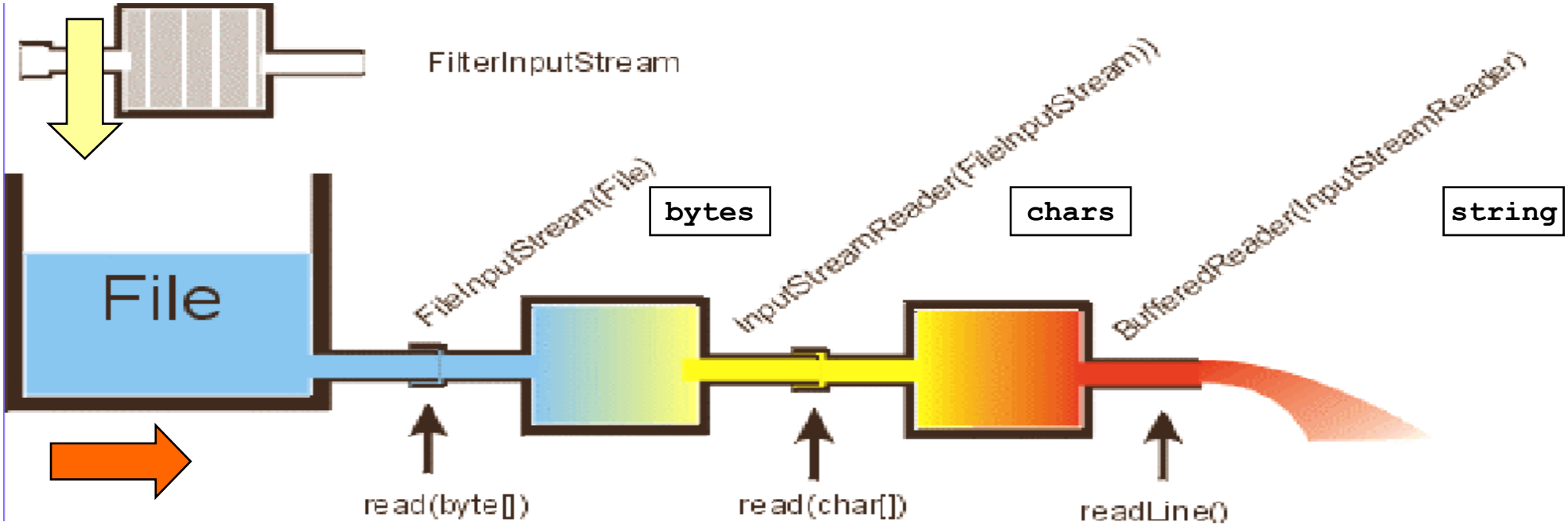
- Principais implementações



I/O em Java

- Buffer e Filtro
  - Exemplo detalhado de uso de filtros e buffer para ler uma linha de um arquivo

É importante observar que o dados são transformados



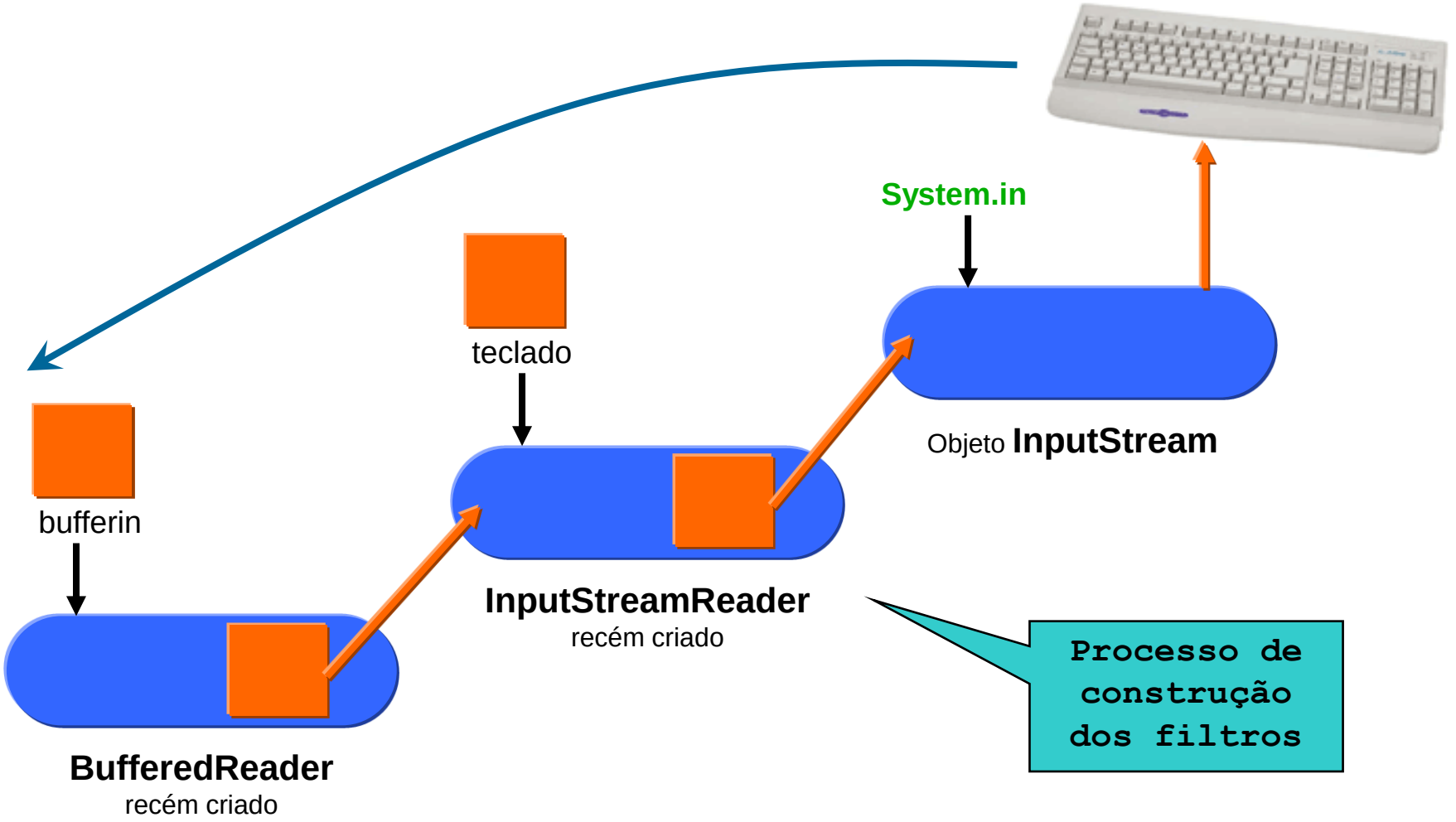

```
// objeto do tipo File
File tanque = new File(" agua.bin ");

// referência FileInputStream
// cano conectado no tanque
FileInputStream cano =
                new FileInputStream(tanque);
```

```
// filtro chf conectado no cano
InputStreamReader chf = new InputStreamReader(cano);
// filtro br conectado no chf
BufferedReader br = new BufferedReader (chf);
// lê linha de texto a de br
String linha = br.readLine();
```

I/O em Java

Vamos implementar juntos..

- Exemplo de Leitura do teclado utilizando *Buffer*
  - Vamos usando o *Reader*....ou seja, estamos trabalhando com caracteres



I/O em Java

- Exemplo de Leitura do teclado utilizando *Buffer*
  - Estamos usando o *Reader*

O que o programa está fazendo?

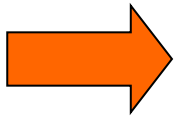
```
public class LeituraTecladoBufferReader {
     public static void main(String[] arguments) {

         // Primeira forma de declaração
         //BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
         InputStreamReader teclado = new InputStreamReader(System.in);
         BufferedReader br = new BufferedReader(teclado);
         String palavra;
// V1.0
         System.out.print("Digite alguma coisa e tecle enter: ");

         try {
             palavra = br.readLine();
             System.out.println("Voce digitou: "+ palavra);
         }
         catch(IOException e) {
             System.out.println("Erro durante a leitura" + e);
         }
     }
  }
```

Vamos implementar juntos..

- V2.0: Alterar o código anterior para solicitar a digitação do texto dentro de um while(true) e que se for digitado “exit” o loop termine (use break).

Exercício

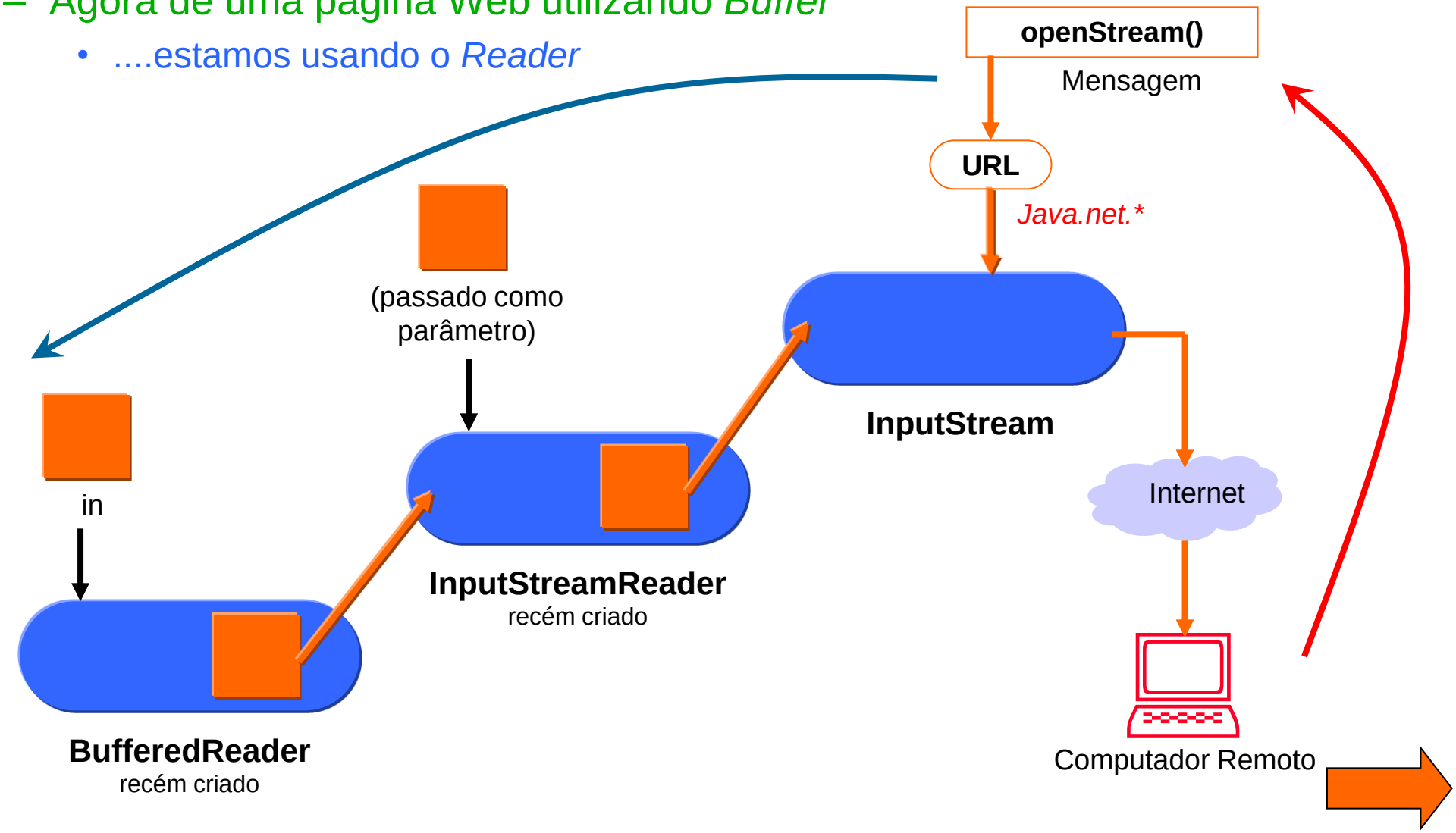
- Desenvolva um programa que leia uma página na Web e a imprima na tela.
  - O programa deve ter as seguintes características
    - 1. Deve utilizar *stream* de caracteres com *Buffer* para obtenção dos dados
    - 2. Deve ler a linha de entrada como *String*
    - 3. Deve criar um objeto da classe URL e abrir uma conexão com a endereço http://www.uvv.br
      - 3.1 Qual método temos que usar para estabelecer a conexão
    - 4. Declare a exceção diretamente no método *main* ou capture com try.
      - Qual devemos utilizar?
    - 5. Utilize como condição de parada o retorno *null* do método *readLine()*
    - 6. Feche o *stream* ao final da computação
    - 7. Após executar o código, redirecione para um arquivo, via comando “>” no prompt



Exercício

Implemente....

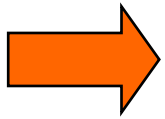
- V3.0: Exemplo de Leitura..
  - Agora de uma página Web utilizando *Buffer*
    - ....estamos usando o *Reader*



Parte II

I/O em Java

- Blz....
  - Agora que entendemos “essas coisas”...
  - ....podemos conversar sobre **manipulação de arquivos..**
  - E sobre ***stream*** **de entrada e saída de arquivos.....**



I/O em Java

- Classe ***File***: Características importantes!!!

⇨ Um objeto *File* representa uma referência de arquivo ou pasta

⇨ É apenas uma abstração:

– Importante: A existência de um objeto File não significa a existência de um arquivo ou diretório

⇨ Contém métodos para:

– Testar a existência de arquivos, para definir permissões (nos SO onde for aplicável), para apagar arquivos, criar diretórios, listar o conteúdo de diretórios, etc.

⇨ Notação multiplataforma

– Prefixo, sequência de strings, File.separator

I/O em Java

- Alguns métodos da classe *File*

  - *String getAbsolutePath()*
  - *String getParent(): retorna o diretório (objeto File) pai*
  - *boolean exists()*
  - *boolean isFile()*
  - *boolean isDirectory()*
  - *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[] list(): retorna lista de arquivos contido no diretório*

I/O em Java

- Trabalhando com **stream de caracteres para entrada e saída** de arquivos e utilização da classe *File*

```
1  import java.io.*;
2
3  public class CopiaUsandoFile {
4      public static void main(String[] args) throws IOException {
5
6          File inputFile = new File("origem.txt");
7          File outputFile = new File("destinoCopiaUsandoFile.txt");
8
9          FileReader in = new FileReader(inputFile);
10         FileWriter out = new FileWriter(outputFile);
11
12         if (outputFile.exists())
13             outputFile.delete();
14
15         int c;
16         while((c = in.read()) != -1)
17             out.write(c);
18
19         in.close();
20         out.close();
21     }
22 }
```

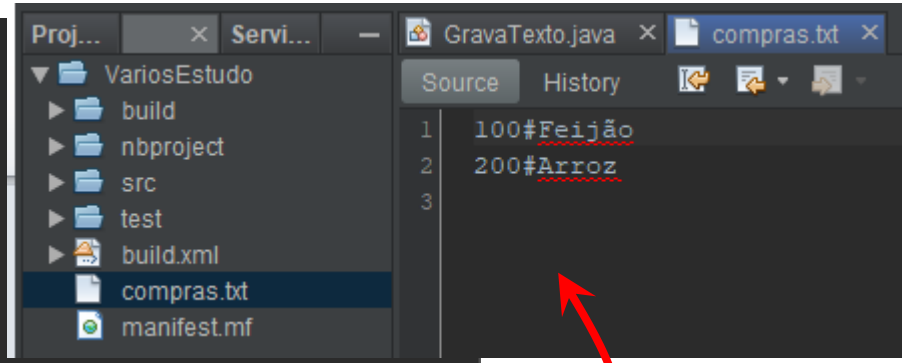
O que o programa está fazendo?

I/O em Java

Resumo importante!!!

Lembre-se: o controle sobre toda a formatação do arquivo (separadores, quebras de linha, etc) é tarefa do programador do sistema

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class GravaTexto{
    public static void main( String[] args) {
        try{
            File file = new File("compras.txt");
            FileWriter fw = new FileWriter(file);
            BufferedWriter bw = new BufferedWriter(fw);
            bw.write("100"); bw.write("#"); bw.write("Feijão"); bw.write("\n");
            bw.write("200"); bw.write("#"); bw.write("Arroz"); bw.write("\n");
            bw.close();
        } catch( IOException e ) {
            e.printStackTrace();
        }
    }
}
```



Escrita formatada do texto

I/O em Java

Resumo importante!!!

Lembre-se: o controle sobre toda a formatação do arquivo (separadores, quebras de linha, etc) é tarefa do programador do sistema

- Para recuperar o texto formatado pode-se usar o método split() ou a classe StringTokenizer.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class LeTexto{
    public static void main( String[] args) {
        try{
        File file = new File("compras.txt");
        FileReader fr = new FileReader(file);
        BufferedReader br= new BufferedReader(fr);
        String linha = null;
        while( (linha = br.readLine()) != null) {
            String[] v = linha.split("#");
            for ( String dado : v )
                System.out.print(dado + " ");
            System.out.println();
        }

        br.close();
        } catch( IOException e ) {
            e.printStackTrace();
        }
    }
}
```

```
run:
100 Feijão
200 Arroz
```

Leitura formatada do texto

I/O em Java

- Blz, pra fechar!!..
  - Lembra que um *stream* é um objeto que transporta dados de um lugar para outro de uma origem para o programa Java ou do programa Java para o destino

  - Logo podemos ter o ***Streams* de objetos**
    - Permite que os dados sejam representados como parte de um objeto
    - Tratam da persistência e recuperação de objetos como um todo para que um objeto seja salvo em um destino, como um arquivo de disco, por exemplo.

Como assim??

I/O em Java

- Serializando Objetos – Qual é a idéia...

Para que um objeto seja salvo em um destino....

- Como um arquivo de disco, por exemplo

....ele precisa ser convertido para a forma serial proprietária ou customizada...

- Dados seriais são enviados um de cada vez, como um fileira de carros em uma linha de montagem.

Um objeto serializado é um grafo que inclui dados da classe e todas as suas dependências que podem ser persistidas

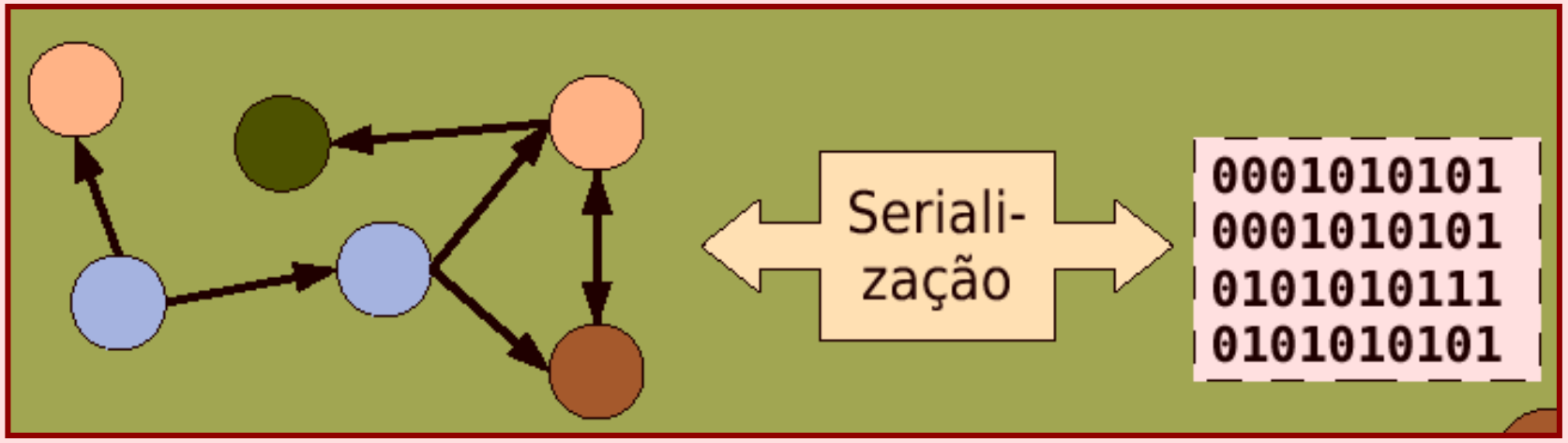
- Se a classe ou suas dependências mudar, o formato usado na serialização mudará e os novos objetos serão incompatíveis com os antigos
  - Não será mais possível recuperar arquivos gravados com a versão antiga

I/O em Java

- Serializando Objetos – Qual é a idéia...

➡ Quando um objeto é salvo em um *stream* de forma serial,

- Todos os objetos que podem ser persistido aos quais ele contém referencia também são salvos



➡ Um objeto que não é serializado não é persistente...

- ....ou seja, deixa de existir após a execução do programa

Como usar isso? ➡

I/O em Java

- 1. Serializando Objetos – Declarando que pode persistir...

➡ Um objeto indica que pode ser serializado....

- ....Por meio da implementação da interface ***Serializable***

➡ Difere das outras interfaces...

- ....Pois não contém métodos que precisam ser incluídos nas classes que o implementam

➡ A única finalidade de interface *Serializable* é....

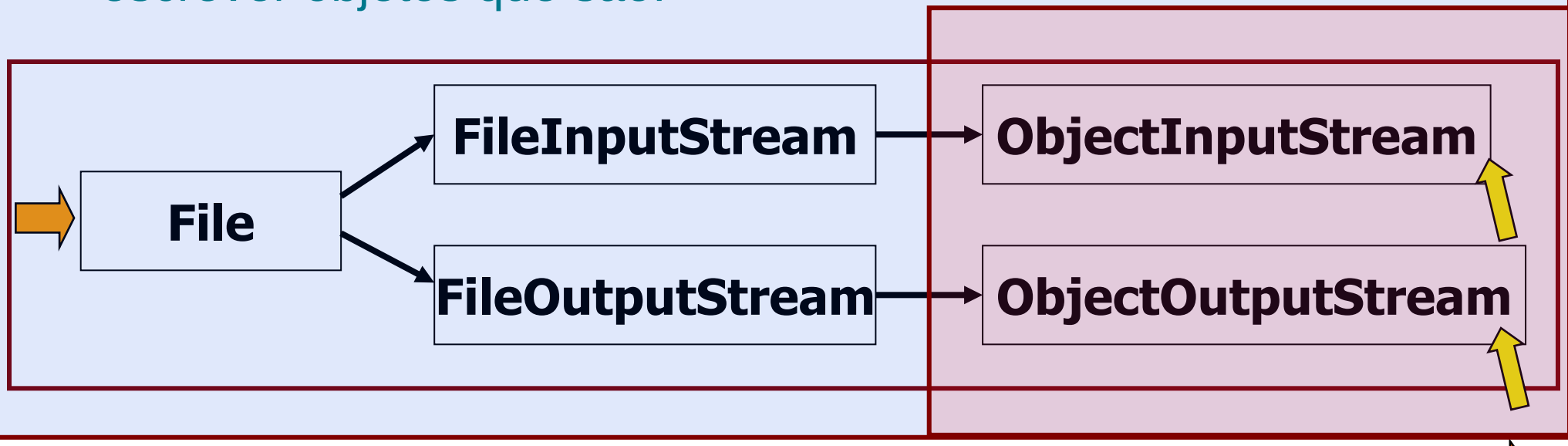
- ....Indicar que os objetos da classe podem ser armazenados e recuperados de forma serial

```
public class Info implements Serializable {
```

I/O em Java - Arquivos e fluxos

- 2. Serializando Objetos – *Streams* Entrada/Saída de objetos

Para trabalhar com entrada e saída serializadas de informações em arquivo...é necessário utilizar as classes envolvidas para ler e escrever objetos que são:



I/O em Java - Arquivos e fluxos

- Trabalhando com Serialização de Objetos – Saída de objetos

➡ Um objeto é escrito em um *stream* por meio da classe ***ObjetctOutputStream***

- Para criar um *stream* de saída para arquivo e um *stream* de saída de objeto associado devemos fazer:

➡ ***FileOutputStream disco = new FileOutputStream (“ObjetoSalvo.dat”);***
ObjectOutputStream obj = new ObjectOutputStream(disco);

- O *stream* de saída de objeto criado nesse exemplo é o *obj*.
- Os métodos da classe *obj* podem ser usados para gravar objetos serializáveis e outras informações em um arquivo chamado *ObjetoSalvo.dat*

➡ Pode-se escrever um objeto utilizando o método *writeObject(Objeto)*

➡ ***obj.writeObject(DadosUsuario)***

– Onde *DadoUsuario* precisa ser declarada como serializável.
- Objeto deve implementar a **`interface java.io.Serializable`**

I/O em Java - Arquivos e fluxos

- Serializando Objetos – Saída de objetos
  - ObjectOutputStream
    - Importante observar a assinatura do método writeObject:

```
public final void writeObject(Object obj)
                                throws IOException
```

Anúncio de exceção, Uso do try\catch no código

I/O em Java - Arquivos e fluxos

- Serializando Objetos – Entrada de objetos

➡ Um objeto é lido de um *stream* usando a classe ***ObjetctInputStream***

- Para criar um *stream* de entrada de arquivo e um *stream* de entrada de objeto associado devemos fazer:

➡ ***FileInputStream disco = new FileInputStream ("ObjetoSalvo.dat");***
ObjectInputStream obj = new ObjectInputStream(disco);

- O *stream* de entrada de objeto criado nesse exemplo é o *obj*.
- Essa *stream* de entrada de objeto é configurada para ler de um objeto que está armazenado em um arquivo chamado *ObjetoSalvo.dat*

➡ Um objeto pode ser lido do arquivo por meio do método *readObject( )*

➡ ***ClasseSerializavel nomeClasse = (ClasseSerializavel) obj.readObject( )***

– Importante observar a realização do *typecast* do objeto recuperado
  - Isto torna possível reconhecer a estrutura do objeto que está sendo lido.

I/O em Java - Arquivos e fluxos

- Serializando Objetos – Entrada de objetos
  - ObjectInputStream
    - Importante observar a assinatura do método readObject:

```
public final Object readObject()
                    throws ClassNotFoundException,
                           IOException
```

Anúncio de exceção, Uso do try\catch no código

I/O em Java - Arquivos e fluxos

- Serializando o objeto - Modelagem / Definição

```
import java.io.Serializable;
import java.util.Date;

/**...4 lines */
class Message implements Serializable {
    int id;
    String from, to;
    Date when;

    void writeMessage(String inFrom, String inTo, Date inWhen) {
        id = inFrom.length();
        to = inTo;
        from = inFrom;
        when = inWhen;
    }
}
```

O que o programa está modelando?

I/O em Java - Arquivos e fluxos

- Serializando o objeto – Saída

O que o programa está fazendo?

```
public class ObjectToDisk {
    public static void main(String[] arguments) {
        Message mess = new Message();
        String author = "Vinicius Rosalen, Brasil";
        String recipient = "Alunos da UVV , UVV - Boa Vista";
        Date now = new Date();

        mess.writeMessage(author, recipient, now);

        try {
            FileOutputStream fo = new FileOutputStream("Message.obj");
            ObjectOutputStream oo = new ObjectOutputStream(fo);
            oo.writeObject(mess);
            oo.close();
            System.out.println("Objeto criado com sucesso.");
        }
        catch (IOException e) {
            System.out.println("Error -- " + e.toString());
        }
    }
}
```

I/O em Java - Arquivos e fluxos

- Recuperando o objeto serializado – Entrada

O que o programa está fazendo?

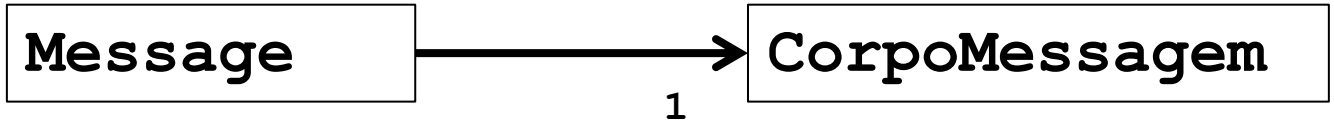
```
public class ObjectFromDisk {

    public static void main(String[] arguments) {
        try {
            FileInputStream fi = new FileInputStream("message.obj");
            ObjectInputStream oi = new ObjectInputStream(fi);

            Message mess = (Message) oi.readObject();
            System.out.println("Remetente: " + mess.from);
            System.out.println("Para: " + mess.to);
            System.out.println("Data: " + mess.when + "\n");
            oi.close();
        }
        catch (Exception e) {
            System.out.println("Error -- " + e.toString());
        }
    }
}
```

I/O em Java - Arquivos e fluxos

- Serializando o objeto – "Exemplo Mensagem"

```
Vamos implementar juntos o
exemplo anterior..
```

- V2.0: Alterar o código anterior para ter uma classe chamada "CorpoMessagem" que tem somente um atributo "String texto" e que se relaciona da seguinte forma:

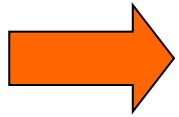


  - Salve e recuperar o objeto Messagem com esse relacionamento de agregação

Exercícios

- Blz... Agora é hora de exercitar.....

- Tente resolver ou analisar os seguintes problemas..
  - Em dupla
  - Apresentar ao professor no final da aula



Exercício

- Serializando Objetos – Exercício (Parte A e B)
  - Criar 3 objetos, gravar em um arquivo, e recuperá-los, mostrando na tela
- (Parte A)
  - Modificar a classe abaixo para que ele possa ser serializada.

```
class PersistenciaDeObjetos_Pessoa_O {
  private int numero;
  private String nome, sobrenome;

  public PersistenciaDeObjetos_Pessoa_O (String nome, String sobrenome, int numero) {
     if (nome == null || sobrenome == null)
        throw new NullPointerException();
     this.nome = nome;
     this.sobrenome = sobrenome;
     this.numero = numero;
  }
}
```

Exercício

- Serializando Objetos – Exercício (Parte A e B)
  - Criar 3 objetos, gravar em um arquivo, e recuperá-los, mostrando na tela
- (Parte B)
  - Para facilitar, crie uma classe programa onde todas as variáveis serão declaradas dentro do método *main()*
    - ***public static void main(String args[])***
  - Crie para escrita em arquivo as seguintes variáveis
    - ***File outfile;***
    - ***FileOutputStream outstream;***
    - ***ObjectOutputStream out;***
  - Crie para leitura de arquivo as seguintes variáveis
    - ***File infile;***
    - ***FileInputStream instream;***
    - ***ObjectInputStream in;***
  - O nome do arquivo para leitura e escrita será "*Objeto.dat*"
  - Use métodos da classe File para verificar, deletar e criar o arquivo "*Objeto.dat*"
  - Não esquecer de utilizar o *close()* para fechar os *streams*

Exercício

- Faça um algoritmo que recebe uma Array de números inteiros e salve em um arquivo “impar.obj”, se o número for impar e em um arquivo “par.obj”, se o número for par.

```
=== SALVANDO AS INFORMAÇÕES ===
Salvando IMPAR: 1
Salvando PAR: 2
Salvando IMPAR: 3
Salvando PAR: 4
Salvando IMPAR: 5
Salvando PAR: 6
Salvando IMPAR: 7
```

- Crie a classe “Dado” para representar a informação computada (o número par ou impar) e serialize esses objetos nos arquivos definidos para cada um.

- Faça um outro algoritmo que recupere esse objeto “Dado”, deserializando e mostrando o resultado persistido nos arquivos.

```
=== RECUPERANDO AS INFORMAÇÕES ===
Recuperando PAR
2
4
6
Recuperando IMPAR
1
3
5
7
```

- Dica: Ao salvar vários dados no arquivo lembre de sinalizar o EoF com null
  - outPar.writeObject(null);
  - outImpar.writeObject(null);