
*Helder da Rocha*
*www.argonavis.com.br*

# Assuntos abordados

- Pacote java.io
- A classe File
- Entrada e saída
  - InputStream e OutputStream
  - Readers e Writers
- Arquivo de acesso aleatório
- Serialização básica
  - ObjectOutputStream e ObjectInputStream

# O pacote java.io

- Oferece abstrações que permitem ao programador lidar com arquivos, diretórios e seus dados de uma maneira independente de plataforma
  - File, RandomAccessFile
- Oferecem 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

# A classe File

- Usado para representar o sistema de arquivos
  - É apenas uma abstração: a existência de um objeto File não significa a existência de um arquivo
  - File pode representar arquivos e diretórios
  - Contém métodos para testar a existência de arquivos, para definir permissões (nos S.O.s onde for aplicável), para apagar arquivos, criar diretórios, listar o conteúdo de diretórios, etc.
- Consulte a documentação para java.io.File e veja os métodos disponíveis

# File: exemplo

```
File diretorio = new File("c:\tmp\cesto");

diretorio.mkdir(); // cria, se possível

File arquivo = new File("lixo.txt", diretorio);

FileOutputStream out =
            new FileOutputStream(arquivo);

out.write(new byte[]{'l','i','x','o'});
// se arquivo não existe, tenta criar
```

# Entrada e Saída

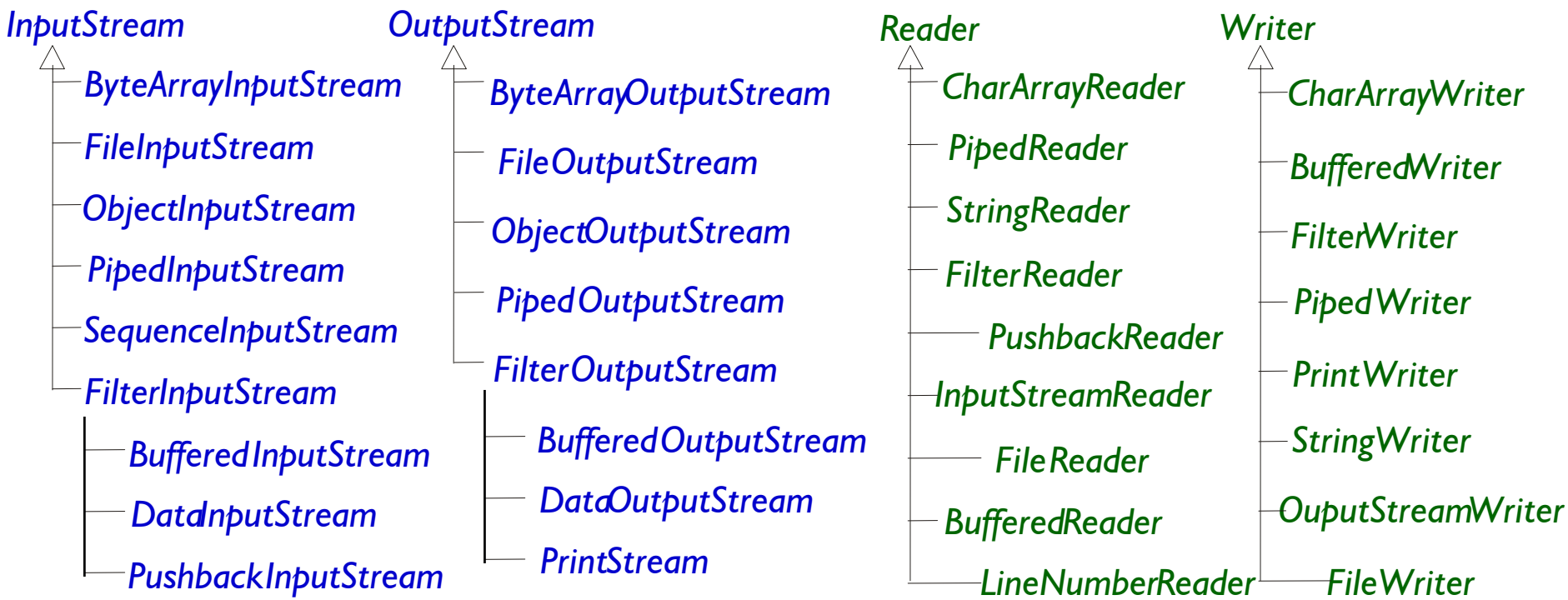
- Fontes de E/S
  - arquivos
  - conexões de rede
  - console (teclado / tela)
  - memória
- Formas diferentes de ler/escrever
  - seqüencialmente / aleatoriamente
  - como bytes / como caracteres
  - linha por linha / palavra por palavra

# API Java para I/O

- API Java lida diretamente com esses problemas
  - classes para lidar com diferentes fontes e destinos de dados
  - classes para filtrar dados recebidos por essas fontes ou enviados para esses destinos

# Principais classes e interfaces

- Dois grupos:
  - e/s de bytes: InputStream e OutputStream
  - e/s de chars: Reader e Writer



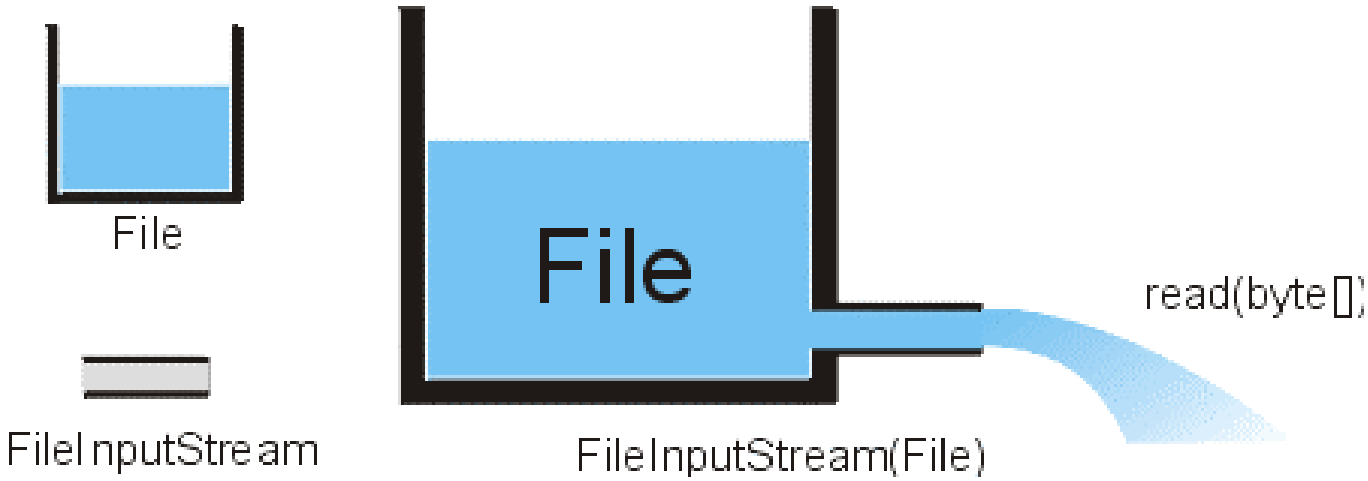
# E/S de bytes

- *InputStream*
  - *Classe genérica (abstrata) para lidar com fluxos de bytes de entrada: método read()*
- *OutputStream*
  - *Classe genérica (abstrata) para lidar com fluxos de bytes de saída: método write()*
- *Principais implementações*
  - *FileInputStream, ByteArrayInputStream, PipedInputStream, FilterOutputStream*
  - *FileOutputStream, ByteArrayOutputStream, PipedOutputStream, FilterOutputStream*

# FilterStreams

- *FilterInputStream: recebe fonte de bytes e lê dados filtrados*
  - *DataInputStream*
  - *BufferredInputStream*
- *FilterOutputStream: recebe destino de bytes e escreve dados via filtro*
  - *DataOutputStream*
  - *BufferedOutputStream*
  - *PrintStream (classe que implementa println)*

# Stream fonte (arquivo)




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

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

// lê um byte a partir do cano
byte octeto = cano.read();
```

# Filtro para ler char

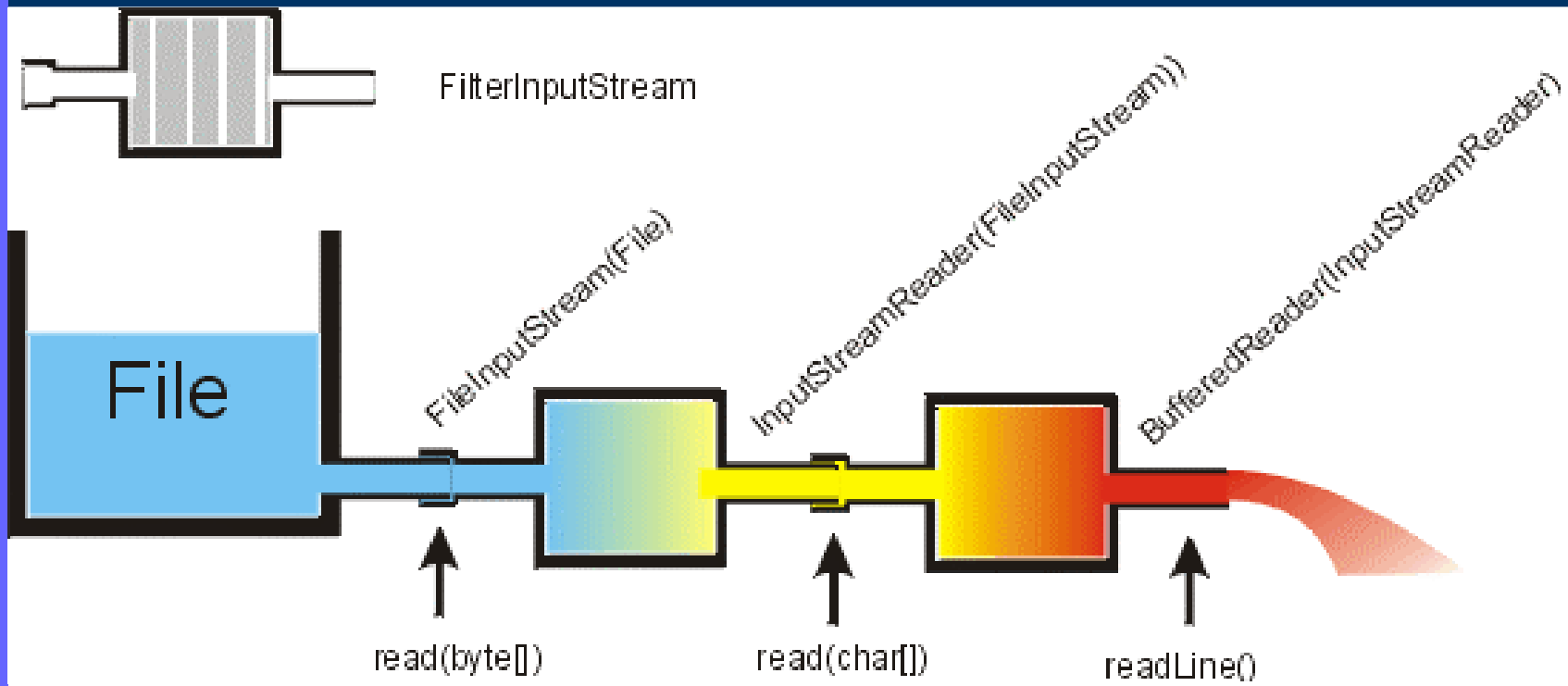
```
// objeto do tipo File
File tanque = new File("agua.txt");

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

// filtro chf conectado no cano
InputStreamReader chf =
             new InputStreamReader(cano);

// lê um char a partir do filtro chf
char letra = chf.read();
```

# Filtro para ler linha



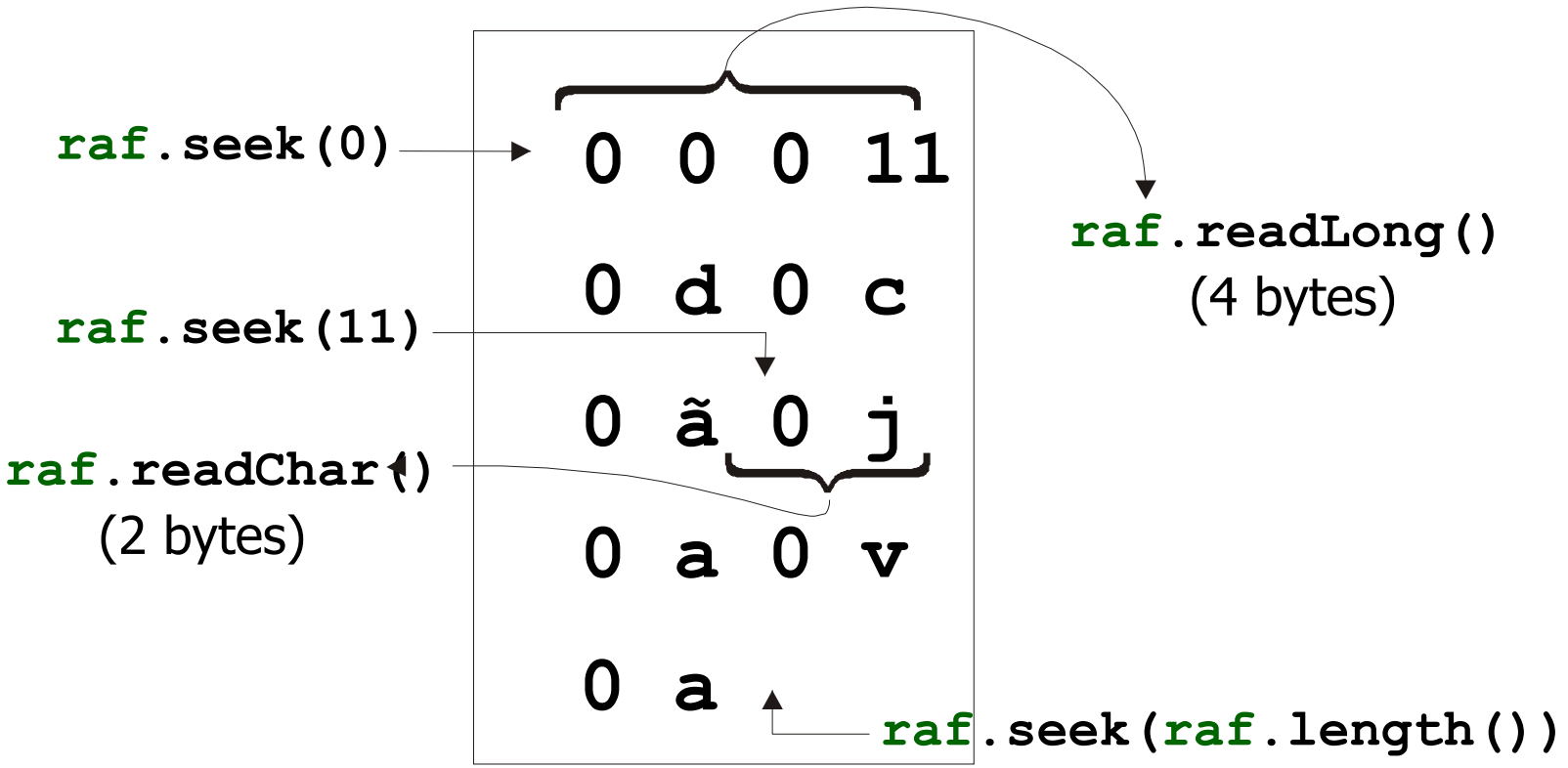

```
// filtro chf conectado no cano
InputStreamReader chf = new InputStreamReader(cano);
// filtro br conectado no chf
BufferedReader br = new BufferedReader (chf);
// lê linha de texto a de br
String linha = br.readLine();
```

# RandomAccessFile

- Classe "alienígena": não faz parte da hierarquia de fluxos de dados do java.io
  - Implementa interfaces DataOutput e DataInput
  - Mistura de File com streams
- Oferece acesso aleatório a um arquivo através de um ponteiro
- Métodos (DataOutput e DataInput) tratam da leitura e escrita de Strings e tipos primitivos
  - void seek(long)
  - readInt(), readBytes(), readUTF(), ...
  - writeInt(), writeBytes(), writeUTF(), ...

# RandomAccessFile

```
RandomAccessFile raf =
        new RandomAccessFile ("arquivo.dat", "rw");
```



# E/S de caracteres

- *Reader*
  - *Classe abstrata para lidar com fluxos de caracteres de entrada: método read() lê um caractere (16 bits) por vez*
- *Writer*
  - *Classe abstrata para lidar com fluxos de bytes de saída: método write() grava um caractere (16 bits) por vez*
- *Principais implementações*
  - *FileWriter, CharArrayWriter, PipedWriter, FilterWriter, BufferedWriter, OutputStreamWriter, PrintWriter*
  - *FileReader, CharArrayReader, PipedReader, FilterReader, BufferedReader, InputStreamReader*

# Exemplo (Livro TIJ*) (1)

```
//: c11:IOStreamDemo.java
import java.io.*;
public class IOStreamDemo {
  public static void main(String[] args)
                              throws IOException {
    // 1. Leitura de dados em linhas:
    BufferedReader in =
      new BufferedReader(
        new FileReader("arquivo.txt"));
    String s, s2 = new String();
    while((s = in.readLine())!= null)
      s2 += s + "\n";
    in.close();
  }
}
```

* Bruce Eckel, "Thinking in Java, 2nd. ed."

# Exemplo (Livro TIJ) (2)

```
// 1b. Leitura de entrada padrão
BufferedReader stdin =
  new BufferedReader(
    new InputStreamReader(System.in));
System.out.print("Digite uma linha:");
System.out.println(stdin.readLine());

// 2. Leitura da memória
StringReader in2 = new StringReader(s2);
int c;
while((c = in2.read()) != -1)
  System.out.print((char)c);
```

# Exemplo (Livro TIJ) (3)

```
// 4. Gravação em disco
try {
  BufferedReader in4 =
    new BufferedReader(
      new StringReader(s2));
  PrintWriter out1 =
    new PrintWriter(
      new BufferedWriter(
        new FileWriter("IODemo.out")));
  int lineCount = 1;
  while((s = in4.readLine()) != null )
    out1.println(lineCount++ + ": " + s);
  out1.close();
} catch(EOFException e) {
  System.err.println("End of stream");
}
```

# Exceptions

- A maior parte das operações de E/S provoca exceções que correspondem ou são subclasses de IOException
  - EOFException
  - FileNotFoundException
  - StreamCorruptedException
- Para executar operações de E/S é preciso, portanto, ou capturar IOException ou repassar a exceção através de declarações throws nos métodos causadores

# ZIP e JAR

- Os pacotes *java.util.zip* e *java.util.jar* permitem comprimir dados e colecionar arquivos mantendo intactas as estruturas de diretórios
- Vantagens
  - Menor tamanho: maior eficiência de E/S e menor espaço em disco
  - Menos arquivos para transferir pela rede (também maior eficiência de E/S)
- Use ZIP e JAR para coleções de arquivos
- Use GZIP para arquivos individuais e para reduzir tamanho de dados enviados pela rede

# Serialização

- *Java permite a gravação direta de objetos em disco ou seu envio através da rede*
- *Para isto, o objeto deve declarar que implementa a interface java.io Serializable*
- *Poderá, então*
  - *ser gravado em qualquer stream usando o método writeObject()*
  - *ser recuperado de qualquer stream usando o método readObject()*

# Gravação de objetos

```
ObjectOutputStream out =
  new ObjectOutputStream(
    new FileOutputStream(armario));

Arco a = new Arco();
Flecha f = new Flecha();

// grava objeto Arco em armario
out.writeObject(a);

// grava objeto flecha em armario
out.writeObject(f);
```

# Leitura de objetos

```
ObjectInputStream in =
   new ObjectInputStream(
     new FileInputStream(armario));

// recupera os dois objetos
// método retorna Object (requer cast)
Arco primeiro = (Arco)in.readObject();
Flecha segundo = (Flecha)in.readObject();
```

# Exercícios

- *1. Faça um programa que leia um arquivo XML ou HTML e arranque todos os tags. Imprima na saída padrão.*
- *2. Crie uma classe RepositorioDadosArquivo que mantenha arquivos armazenados em três diretorios:*
  - *assuntos/*
  - *agentes/*
  - *publicacoes/*

  *Cada diretório deverá armazenar um arquivo por registro. O arquivo deve ter o codigo do registro e os dados devem estar separados por virgulas. Use '\\' para representar uma contra-barra e '\,' para representar uma vírgula.*