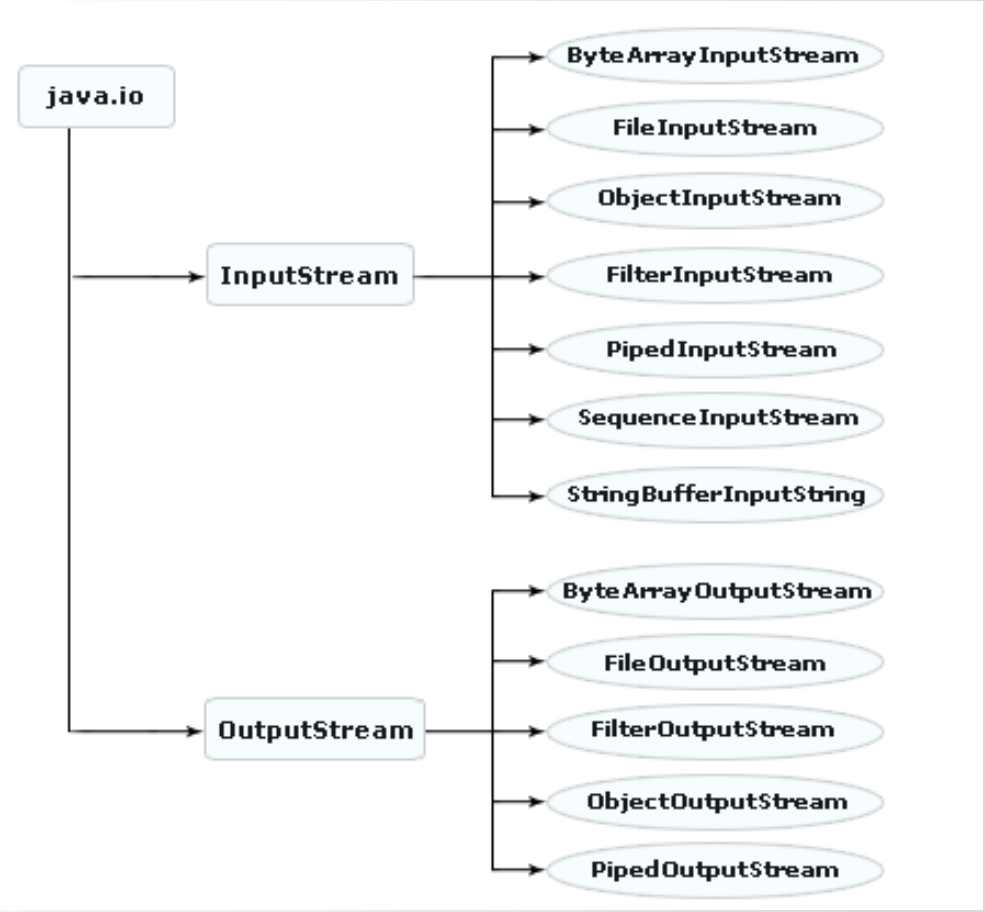
Entrada e Saída

Introdução

- Input e Output fazem parte do pacote java.io
- Esse pacote tem uma quantidade grande de classes que dão suporte a operações de entrada e saída
- As classes básicas são InputStream e OutputStream

Hierarquia de classes

- Veja as diversas classes do pacote



Tipos de Streams

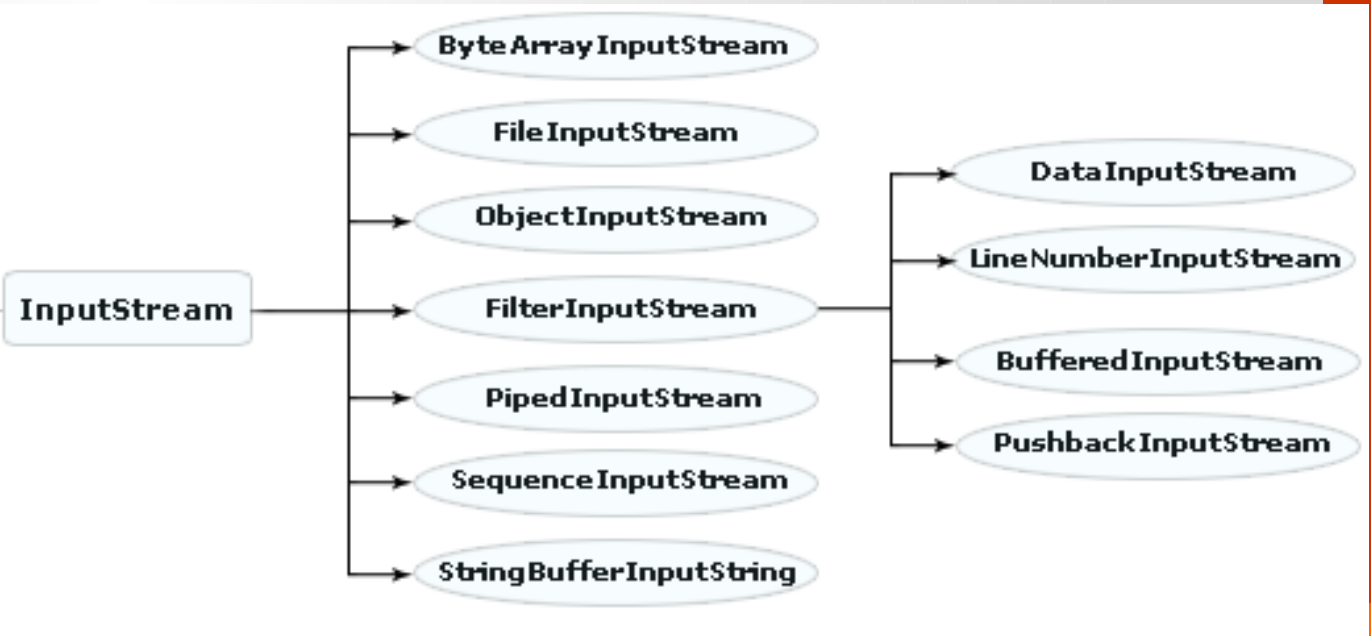
- Existem dois tipos de Streams: binário e texto
- No primeiro a gente pode transferir bytes (arquivos, por exemplo), no segundo, caracteres

InputStream

- InputStream é usada para ler dados como bytes de uma fonte qualquer (arquivo, String, memória)
- É uma classe abstrata
- Você deve fechar seu Stream após o uso, ou esperar que seja coletado pelo Garbage collector

Hierarquia de InputStream

- Você deve usar uma dessas subclasses
- Cada uma tem um propósito diferente

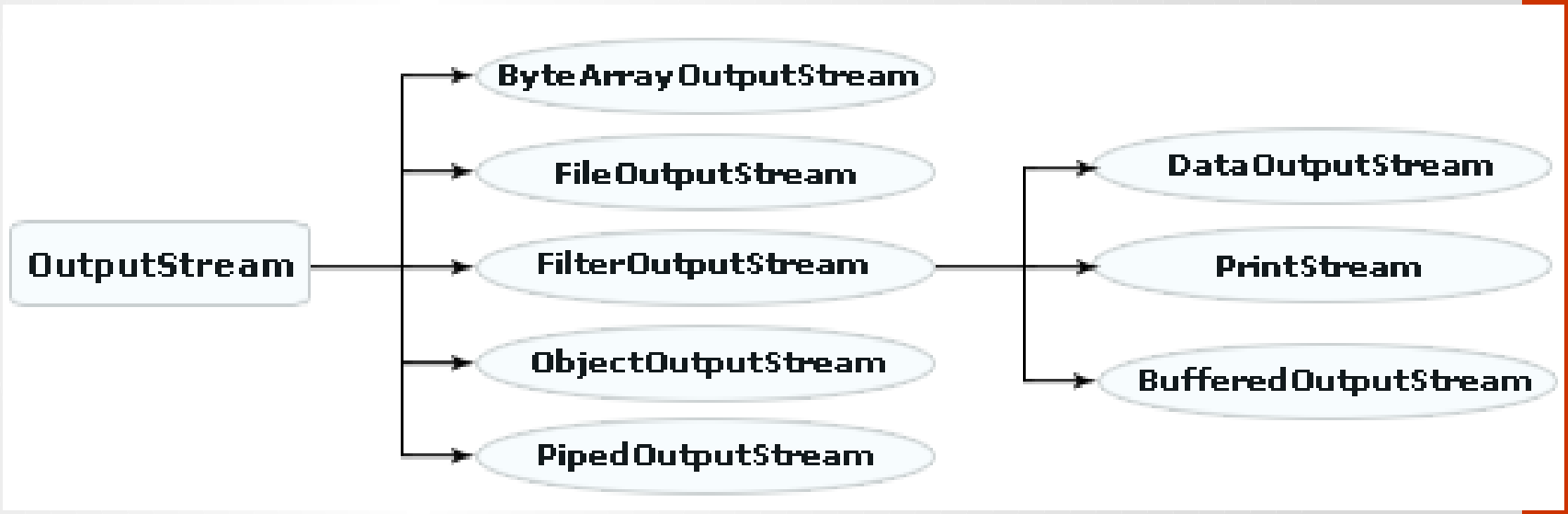


OutputStream

- OutputStream é similar a sua colega
- É usada para escrever bytes em alguma fonte
- Você pode ler do disco e gravar na memória, ler um objeto e gravar no disco...
- Todas as combinações são válidas, você faz de acordo com a conveniência

Hierarquia

- Mais uma vez você só pode usar uma das subclasses de OutputStream



File

- Essa classe trabalha com os arquivos do computador de forma independente
- Dessa forma, você pode escrever código que manipula arquivos independentemente da plataforma
- Um objeto do tipo File pode representar tanto um arquivo como um diretório
- Quando ele é criado, o SO não checa a existência efetiva do arquivo/diretório

Principais métodos

- f.exists() – true se o arquivo existir
- f.isFile() – true se é um arquivo
- f.isDirectory() – true se é um diretório
- f.getName() – nome do arquivo/diretório
- f.length() – número de bytes de um arquivo
- f.getPath() – nome do caminho
- f.delete() – apaga
- f.renameTo(f2) – renomeia para f2
- f.createNewFile() – cria o arquivo (pode disparar IOException)

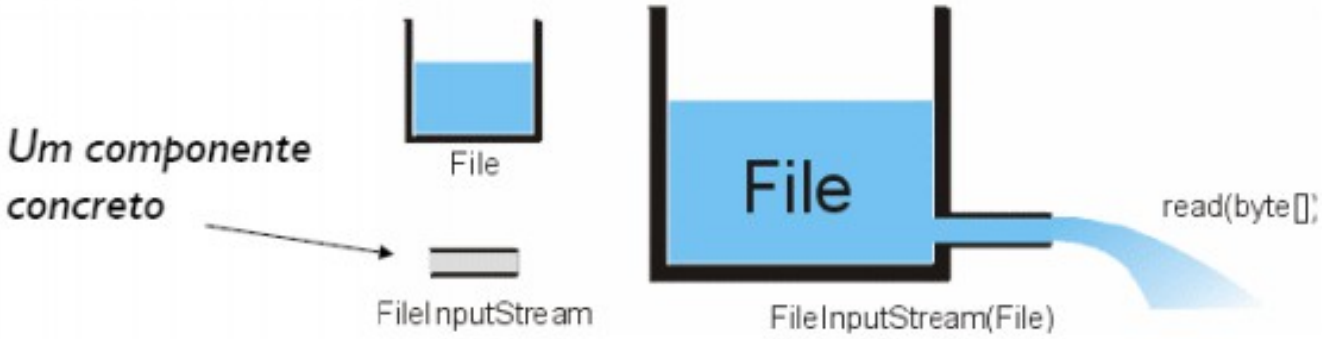
Exemplo

```
package net.stream;

import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;

public class TrabalhandoComArquivos {
  public static void main(String[] args) throws Exception {
    File arquivo = new File("c:/install.log");
    System.out.println(“Existe: ” + arquivo.exists());
    System.out.println(“Nome do arquivo: ” + arquivo.getName());
    System.out.println(“Tamanho em bytes: “ + arquivo.length());
  }
}
//Teste outros métodos!!!
```

Usando Streams

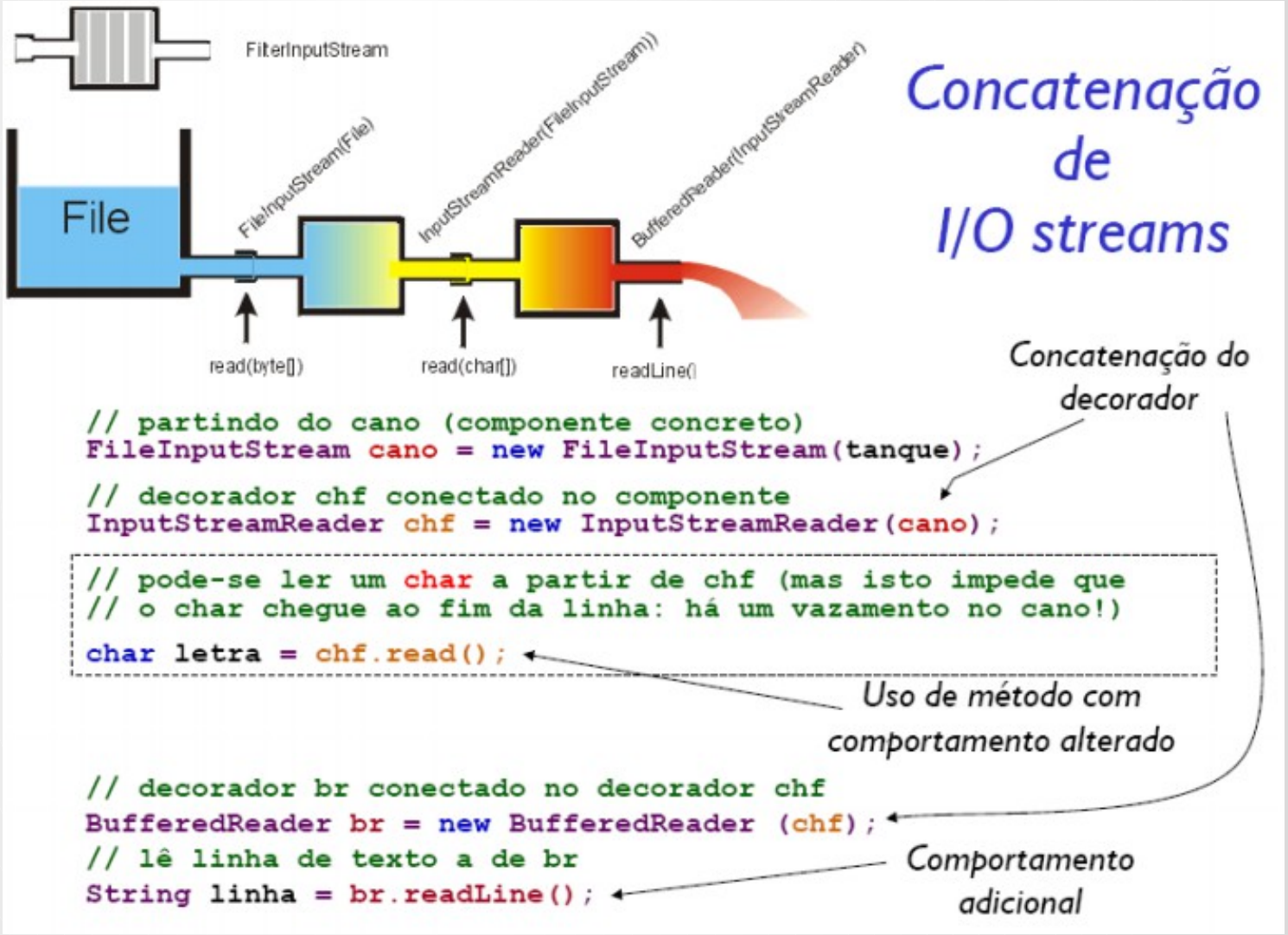



```
// objeto do tipo File
File tanque = new File("agua.txt");

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

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

Usando Streams



Lista Arquivos

```
package com.javabasico.entradaesaida;
import java.io.*;
public class ListaArquivos {
  public static void main(String[] args) {
    File diretorio = new File("/Users/marcoreis/Software");
    if (diretorio.isDirectory()) {
      for (String nomeDoArquivo : diretorio.list()) {
            String caminho = diretorio.getPath();
            File arquivo = new File(caminho + "/" + nomeDoArquivo);
            if (arquivo.isFile()) {
                  System.out.print(arquivo.getName() + " - ");
                  long tamanhoEmMB = arquivo.length() / 1024;
                  System.out.println(tamanhoEmMB + "MB");
            }
      }
    }
  }
}
```

Escrevendo arquivos

- Uma operação bastante utilizada no desenvolvimento de software é a criação de arquivos
- Podemos fazer de diversas formas diferentes
- Para tanto, vamos começar com arquivos texto mais simples
- As classes usadas: FileOutputStream e FileInputStream

Escrevendo arquivos

```
package net.stream;

import java.io.File;
import java.io.FileOutputStream;

public class EscrevendoArquivos {
  public static void main(String[] args) {
        try {
            File f = new File("c:/NovoArquivo.txt");
            FileOutputStream fo = new FileOutputStream(f);
            String texto = "Este é o texto que vamos gravar no arquivo";
            texto = texto + "\nEsta é a segunda linha";
            fo.write(texto.getBytes());
            fo.close();
            System.out.println("Arquivo gravado com sucesso");
        } catch (Exception e) {
     e.printStackTrace();
    }
  }
}
```

Detalhamento

- Tratamento de exceção obrigatório
- Arquivo a ser criado/atualizado
- OutputStream para arquivos
- Texto a ser gravado
- Gravação do texto na saída (NovoArquivo.txt). Atenção: em geral, os bytes são transferidos em forma de array
- Não podemos esquecer de fechar o recurso
- Mensagem de confirmação
- Tratamento de exceção é extremamente indicado

Lendo arquivos

- Para ler um arquivo (ou qualquer outra fonte de dados) através da stream, devemos recuperar uma linha de cada vez, ou seja, fazer um loop até chegar no fim dos dados
- Temos basicamente 2 métodos para trabalhar com streams: read() e write()
- Existem diversas variações para as mais variadas situações. No fim das contas o princípio é o mesmo: escreve depois lê. Pronto

Lendo arquivos

```
package net.stream;

import java.io.File;
import java.io.FileInputStream;

public class LendoArquivos {
  public static void main(String[] args) {
    try {
        File f = new File("c:/NovoArquivo.txt");
        FileInputStream fi = new FileInputStream(f);
        int i = 0;
        while(i!=-1){
                i = fi.read();
                char c = (char) i;
                System.out.print(c);
        }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Detalhamento

- Ponteiro para o arquivo a ser lido
- InputStream para ler o arquivo
- Vamos ler os bytes do arquivos e armazenar na variável i
- A flag que indica fim do arquivo é i = -1, ou seja, vamos ler enquanto não encontrar esse valor
- Lê um byte de cada vez
- Converte de inteiro para char (int e char são iguais)
- Imprime o caracter

Lendo do buffer

```
package net.stream;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;

public class LendoLinhasInteiras {
  public static void main(String[] args) {
    try {
        FileInputStream fi = new FileInputStream("c:/NovoArquivo.txt");
        InputStreamReader ir = new InputStreamReader(fi);
        BufferedReader br = new BufferedReader(ir);
        String linha;
        while ((linha = br.readLine()) != null) {
                System.out.println(linha);
        }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Detalhes

- InputStream para leitura de arquivos
- O reader funciona como um leitor específico para o buffer, possibilitando a leitura de uma linha inteira. De outra forma, somente byte a byte
- Leitor do buffer
- Variável que vai armazenar as linhas do arquivo
- Fazemos a atribuição e a comparação ao mesmo tempo
- Imprime a linha do arquivo

Serializando objetos

- Em algumas situações precisamos transportar objetos pela rede. Antes disso, o objeto precisa ser gravado no disco (serializado), depois é só enviar por um socket para o computador remoto
- Vamos criar uma classe de teste e serializá-la no disco
- Depois a gente recupera a mesma, preservando seu estado

Classe de teste

```
package net.stream;

import java.io.Serializable;

public class AlgumaClasse implements
Serializable {
  private String nome;
  private String outroNome;
  private String maisOutroNome;

  public String getNome() {
    return nome;
  }

  public void setNome(String nome) {
    this.nome = nome;
  }

  public String getOutroNome() {
    return outroNome;
  }

  public void setOutroNome(String outroNome) {
    this.outroNome = outroNome;
  }

  public String getMaisOutroNome() {
    return maisOutroNome;
  }

  public void setMaisOutroNome(String
maisOutroNome) {
    this.maisOutroNome = maisOutroNome;
  }
}
```

Serializando

```
package net.stream;

import java.io.FileOutputStream;
import java.io.ObjectOutputStream;

public class SerializandoObjetos {
  public static void main(String[] args) {
    try {
          AlgumaClasse a = new AlgumaClasse();
          a.setMaisOutroNome("a");
          a.setNome("b");
          a.setOutroNome("c");
          FileOutputStream fo = new FileOutputStream("c:/classe.tmp");
          ObjectOutputStream ou = new ObjectOutputStream(fo);
          ou.writeObject(a);
          ou.close();
          fo.close();
          System.out.println("Objeto serializado");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Detalhes

- Criamos o objeto e atribuímos valores
- Stream para o arquivo de saída
- Stream para trabalhar com objetos
- Escreve o objeto no arquivo
- Fecha o stream de objetos
- Fecha o stream do arquivo

Lendo do disco

- Agora que gravamos, precisamos recuperar o objeto do disco, não é?
- O núcleo de tudo é o método readObject(), que é bastante sugestivo
- Vamos

Deserializar

```
package net.stream;

import java.io.FileInputStream;
import java.io.ObjectInputStream;

public class DeserializandoObjetos {
  public static void main(String[] args) {
    try {
          FileInputStream fi = new FileInputStream("c:/classe.tmp");
          ObjectInputStream oi = new ObjectInputStream(fi);
          Object o = oi.readObject();
          AlgumaClasse a = (AlgumaClasse) o;
          System.out.println(a.getMaisOutroNome());
          System.out.println(a.getNome());
          System.out.println(a.getOutroNome());
          oi.close();
          fi.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Passo a passo

- InputStream para ler um arquivo do disco
- InputStream para ler um objeto
- Lê o objeto do disco. Atenção: sempre trabalhamos com Object, o que nos obriga a fazer cast de tudo
- Faz o cast para AlgumaClasse
- Imprime os valores do objeto recuperado

PrintWriter

- Classe para impressão rápida de caracteres em um OutputStream, inclusive fazendo formatação
- No nosso exemplo, a saída vai ser um arquivo texto
- Existe outra classe chamada PrintStream que trabalha com bytes

PrintWriter

```
package net.stream;

import java.io.File;
import java.io.PrintWriter;

public class UsandoPrintWriter {
  public static void main(String[] args) {
    try {
          File f = new File("c:/ArquivoPrintWriter.txt");
          PrintWriter p = new PrintWriter(f);
          p.print("primeira linha");
          p.println("segunda linha");
          p.write("terceira linha");
          p.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Exercício

- Elabore uma classe CadastroAluno para armazenar uma lista de objetos alunos em um arquivo texto.
- Ela deve operar os objetos Aluno em uma lista (incluir(aluno), excluir(id), listaTodos(), busca(id) e possuir métodos para gravar() e recuperar() a lista em disco.
- Armazene os arquivo em formato texto e abra com o notepad para verificar.