



Arquivos

Prof. Renato Pimentel

2023/2



Sumário



1 Arquivos



Há várias fontes (entrada) de onde se deseja ler, ou destinos (saída) para onde se deseja gravar ou enviar dados:

- **Arquivos;**
- Memória;
- Teclado, tela, impressora, mouse, etc.

Há várias formas diferentes de ler/escrever dados:

- Sequencialmente/aleatoriamente
- Como bytes, como caracteres
- Linha por linha, palavra por palavra, etc



Como oferecer tais serviços em Java??



A linguagem Java *não* trata dispositivos de entrada e saída (E/S) de forma específica, ou seja, com classes específicas para cada dispositivo; Ao invés disso, Java utiliza um mecanismo genérico que permite tratar E/S de forma uniforme: **Streams de entrada e saída**.

Stream

Um **stream** é um canal por onde trafegam dados entre um processo computacional e uma origem – ou destino – de dados.

Mais detalhes:

<https://docs.oracle.com/javase/8/docs/api/java/io/package-summary.html>



A **ordem** do fluxo de dados, entrada ou saída, é relevante na escolha do *stream* a ser utilizado.

- **Stream de entrada:** para obter informações, uma aplicação abre um *stream* de uma fonte (arquivo, *socket*, memória, etc) e lê os dados desejados;
- **Stream de saída:** Para enviar informações, uma aplicação abre um *stream* para um destino (arquivo, *socket*, memória) e escreve os dados.

Independentemente da fonte/destino e do tipo de informações, os algoritmos para leitura e escrita são basicamente os mesmos.



Algoritmo de leitura:

```
abre um stream
enquanto há informação
    lê informação
fecha o stream
```



Algoritmo de escrita:

```
abre um stream
enquanto há informação
    escreve informação
fecha o stream
```



Pacote `java.io`

O pacote `java.io` contém uma coleção de classes para trabalhar com fluxo de entrada e saída de dados; Estas classes dão suporte aos algoritmos de E/S – ou I/O, de *Input/Output*; As classes são divididas em dois grupos, baseadas no tipo de dados sobre os quais operam:

- `InputStream` e `OutputStream`: E/S de *bytes* – suportam leitura e gravação de 8 bits;
- `Reader` e `Writer`: E/S de caracteres (`char`) – suportam leitura e gravação de caracteres **Unicode** de 16 bits.

Estas classes são abstratas, ou seja, contêm um ou mais métodos para os quais não há definição. Sendo assim, objetos não podem ser criados a partir de classes abstratas.



Entrada de *bytes* – `java.io.InputStream`



Classe `java.io.InputStream`: classe abstrata para lidar com fluxos de *bytes* (**dados binários**, como imagens e sons) de entrada;

<https://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html>

Método básico: `int read()`

Usado na leitura dos dados disponíveis em um *stream*. Note que o retorno do método é um número inteiro entre 0 e 255, indicando o *byte* lido do *stream*, ou:

Caso não haja *bytes* disponíveis para a leitura (final de arquivo), o retorno deste método será `-1`.

Em caso de erro, uma `IOException` é lançada.



Classe `java.io.OutputStream`: classe abstrata para lidar com fluxos de *bytes* (**dados binários**, como imagens e sons) de saída – ou seja, **dados para gravação**;

<https://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.html>

Método básico: `void write()`



Classe `java.io.Reader`: classe abstrata para lidar com fluxos de **caracteres Unicode** de entrada;

<https://docs.oracle.com/javase/8/docs/api/java/io/Reader.html>

Método básico: `int read()`

Lê um caractere de 16 bits por vez, a partir de um *stream*.



Classe `java.io.Writer`: classe abstrata para lidar com fluxos de caracteres **Unicode** de saída;
<https://docs.oracle.com/javase/8/docs/api/java/io/Writer.html>

Método básico: `void write()`

Grava um caractere de 16 bits por vez em um *stream* de saída.



Persistência de dados



Duas abordagens comuns para implementar a persistência de dados:

- Armazenar dados em arquivos texto;
- Usar **serialização**, permitindo gravar objetos em arquivos.

Persistência de dados

Persistência de dados consiste no armazenamento **confiável** e coerente das informações em um sistema de armazenamento de dados.



- Os dados são salvos em arquivos, separados por algum caractere único, como por exemplo `' , '`;
- Um arquivo texto pode ser editado e visualizado facilmente por humanos;
- Simples para fazer intercâmbio de dados entre programas diferentes.

Exemplo: um arquivo chamado `teste.dat`:

```
1 José da Silva,23,7.5
2 Márcia Bastos,20,7.0
3 Carla Pereira,18,8.5
```



Compreende a **criação** do arquivo, o **armazenamento dos dados**, e o **fechamento** do arquivo:

`FileWriter`: Estabelece a conexão com o arquivo. Usado para a saída em um arquivo, baseada em caracteres:

```
FileWriter arq = new FileWriter( nomeArq );
```

`PrintWriter`: Para escrevermos Strings no arquivo, precisamos de um objeto `PrintWriter` associado ao `FileWriter`:

```
PrintWriter out = new PrintWriter( arq );
```

Podemos então usar os métodos `print()` e `println()` da classe `PrintWriter`;

Devemos implementar o código dentro de um bloco `try/catch`, pois exceções podem ser geradas (`IOException`).



`BufferedWriter`: Esta classe permite uma saída bufferizada: Uma operação de saída **não grava imediatamente** os dados no arquivo.

Com o método `flush()`, de tempos em tempos uma quantidade de dados é enviada para o arquivo.



Consiste na recuperação das informações armazenadas em um arquivo, para serem utilizadas por determinado programa:

`FileReader`: Estabelece a conexão com o arquivo. Uma operação de entrada lê um caractere, ou seja, trabalha *com um caractere por vez*.

```
FileReader ent = new FileReader( nomeArq );
```

`BufferedReader`: Entrada bufferizada. Uma operação de entrada lê vários caracteres de uma única vez

```
BufferedReader br = new BufferedReader (ent);
```

Método utilizado para leitura: `br.readLine()`

Este método retorna `null` quando o final do arquivo for atingido.



Exemplo – UsarArquivo.java



```
1 package usararquivos;
2 public class UsarArquivos {
3     public static void main(String[] args) {
4         String nome[] = new String[3];
5         int idade[] = new int[3];
6         double nota[] = new double[3];
7         nome[0] = "José da Silva";
8         nome[1] = "Márcia Bastos";
9         nome[2] = "Carla Pereira";
10        idade[0] = 23;
11        idade[1] = 20;
12        idade[2] = 18;
13        nota[0] = 7.5;
14        nota[1] = 7;
15        nota[2] = 8.5;
16        GerenciamentoArquivos gerente = new GerenciamentoArquivos();
17        gerente.escrita("teste.dat", nome, idade, nota);
18        gerente.leitura("teste.dat");
19    }
20 }
```



Exemplo – GerenciamentoArquivos.java I



```
1 package usararquivos;
2 import java.io.*;
3
4 public class GerenciamentoArquivos {
5     public void escrita(String nomeArq, String[] vet1, int[] vet2,
6         double[] vet3) {
7         try {
8             FileWriter arq = new FileWriter(nomeArq);
9             PrintWriter out = new PrintWriter(arq);
10            for (int i = 0; i < vet1.length; i++) {
11                String linha = vet1[i] + ":" + vet2[i] + ":" + vet3[i];
12                out.println(linha);
13            }
14            out.close();
15        } catch (IOException erro) {
16            System.out.println("Erro na escrita dos dados");
17        }
18    } //fim do método escrita()
19
20    public void leitura(String nomeArq) {
21        try {
```



Exemplo – GerenciamentoArquivos.java II



```
21     FileReader ent = new FileReader(nomeArq);
22     BufferedReader br = new BufferedReader(ent);
23     String linha;
24     String[] campos = null;
25     while ((linha = br.readLine()) != null) {
26         campos = linha.split(":");
27         String nome = campos[0];
28         int idade = Integer.parseInt((campos[1]));
29         double nota = Double.parseDouble(           campos[2].
30     replace(",", ".")) );
31         System.out.println("Nome=" + nome + " Idade=" + idade + "
32     Nota=" + nota);
33     }
34     br.close();
35     } catch (IOException erro) {
36         System.out.println("Erro na leitura dos dados");
37     }
38 } // Fim do método leitura()
39 } // Fim da classe
```



Exemplo – GerenciamentoArquivos.java III





Serialização: Processo de transformar o **estado** de um objeto em uma sequência de *bytes* que representem o valor de seus atributos:

- Obs: métodos e construtores não fazem parte da serialização;
- Após a serialização, é possível gravar o objeto serializado (sequência de *bytes*) em um arquivo, enviá-lo através da rede, etc.



Deserialização: é o processo inverso, de reconstruir um objeto a partir de uma sequência de *bytes* para o mesmo estado que o objeto estava antes de ser serializado:

- Quando os objetos forem recuperados, é preciso recriar as instâncias e reconectá-las da maneira correta.



Como permitir a serialização/deserialização em Java?

Fazendo os objetos implementarem a interface **Serializable** (do pacote `java.io`).

Serializable não tem métodos: serve apenas para indicar que os atributos destes objetos podem ser serializados e deserializados.



Escrita de objetos – serialização I



Passos para gravar/escrever um objeto serializado em um arquivo:

- Criar um objeto `FileOutputStream`:
`FileOutputStream arq = new FileOutputStream(nomeArq);`
- Criar um objeto `ObjectOutputStream`:
`ObjectOutputStream os = new ObjectOutputStream(arq);`
- Gravar o objeto:
`os.writeObject (objeto);`
- Fechar o objeto `ObjectOutputStream`:
`os.close();`



Não esquecer

Para que uma classe seja serializada, ela deve implementar a interface `Serializable`



Leitura de objetos – deserialização



Restauração do estado de um objeto:

- Criar um objeto `FileInputStream`:
`FileInputStream arq = new FileInputStream(nomeArq);`
- Criar um objeto `ObjectInputStream`:
`ObjectInputStream is = new ObjectInputStream(arq);`
- Ler o objeto:
`Medicamento m=(Medicamento) is.readObject();`
- Trabalhar com o objeto:
`System.out.print("Nome: " + m.getNome());`
- Fechar o objeto `ObjectOutputStream`:
`is.close();`



Exemplo – Medicamento.java I



```
1 package serialfarmacia;
2
3 import java.io.Serializable;
4
5 public class Medicamento implements Serializable {
6
7     String nome;
8     double preco;
9
10    public Medicamento() {
11    }
12
13    public Medicamento(String novoNome, double novoPreco) {
14        this.nome = novoNome;
15        this.preco = novoPreco;
16    }
17
18    public void setNome(String novoNome) {
19        this.nome = novoNome;
20    }
```



Exemplo – Medicamento.java II



```
21
22
23    public void setPreco(double novoPreco) {
24        this.preco = novoPreco;
25    }
26
27    public String getNome() {
28        return this.nome;
29    }
30
31    public double getPreco() {
32        return this.preco;
33    }
34
35    public void escreverMedicamento() {
36        System.out.println("Nome" + this.nome);
37        System.out.println("Preco" + this.preco);
38    }
39 }
```



```
1 package serialfarmacia;
2
3 public class TestaFarmaciaSerializacao {
4
5     public static void main(String[] args) {
6         Farmacia ufu = new Farmacia();
7         /* cadastro de medicamentos */
8         Medicamento m = new Medicamento("a", 5.6);
9         ufu.cadastraMedicamento(m);
10        m = new Medicamento("b", 15.6);
11        ufu.cadastraMedicamento(m);
12        m = new Medicamento("c", 25.6);
13        ufu.cadastraMedicamento(m);
14        m = new Medicamento("d", 35.6);
15        ufu.cadastraMedicamento(m);
16        m = new Medicamento("e", 3.6);
17        ufu.cadastraMedicamento(m);
18
19        //Serializa os objetos
```




```
20     ufu.escreverMedicamentos("medicamentos.dat");
21
22     //Deserializa os objetos
23     ufu.lerMedicamentos("medicamentos.dat");
24 }
25
26 }
```



```
1 package serialfarmacia;
2
3 import java.io.*;
4
5 public class Farmacia {
6
7     Medicamento lista[] = new Medicamento[100];
8     int estoque = 0;
9
10    public void cadastraMedicamento(Medicamento m) {
11        lista[estoque] = m;
12        estoque++;
13    }
14
15    public void cadastrMedicamento(String nome, double preco) {
16        Medicamento m = new Medicamento(nome, preco);
17        lista[estoque] = m;
18        estoque++;
19    }
}
```



Exemplo – Farmacia.java II



```
20
21
22
23 //OUTROS MÉTODOS
24 public void escreverMedicamentos() {
25     for (int i = 0; i < estoque; i++) {
26         lista[i].escreverMedicamento();
27     }
28 }
29
30 public void escreverMedicamentos(String nomeArq) {
31     try {
32         FileOutputStream arq = new FileOutputStream(nomeArq);
33         ObjectOutputStream os = new ObjectOutputStream(arq);
34         for (int i = 0; i < estoque; i++) {
35             os.writeObject(lista[i]);
36         }
37         os.close();
38         arq.close();
39     } catch (IOException erro) {
```



Exemplo – Farmacia.java III



```
40         System.out.println("Ocorreu um erro na escrita dos dados" +
41         erro);
42     }
43 } // Fim do método escreverMedicamentos( String )
44
45
46 public void lerMedicamentos(String nomeArq) {
47     try {
48         FileInputStream arq = new FileInputStream(nomeArq);
49         ObjectInputStream is = new ObjectInputStream(arq);
50         for (int i = 0; i < estoque; i++) {
51             Medicamento m = (Medicamento) is.readObject();
52             System.out.print("Nome: " + m.getNome());
53             System.out.println(" Preço: " + m.getPreco());
54         }
55         is.close(); arq.close();
56     } catch (IOException erro) {
57         System.out.println("Ocorreu um erro na escrita dos dados: " +
58         erro);
```



```
58     } catch (ClassNotFoundException erro) {  
59         System.out.println("Ocorreu um erro de leitura no arquivo: " +  
60         erro);  
61     }  
62 } //Fim do método lerMedicamentos()  
63 } //Fim da classe
```



Exercício I



Faça um programa que leia um arquivo de dados contendo 4 inteiros, cada inteiro correspondendo a uma quantidade de votos para um determinado time. Mostre essas informações na tela para o usuário.



- ① Apostila de Java e POO Caelum: disponível em <https://www.caelum.com.br/download/caelum-java-objetos-fj11.pdf> – acesso em: MAI/2017.

Os slides de parte desta seção foram cedidos por Marcelo Z. do Nascimento, FACOM/UFU

LaTeXagem e adaptações: Renato Pimentel, FACOM/UFU