Arquivos

Prof. Renato Pimentel

2024/2

Sumário

Arquivos I

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

Arquivos II

Como oferecer tais serviços em Java??

Stream (fluxo) I

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

Stream (fluxo) II

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.

Stream (fluxo) III

Algoritmo de leitura:

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


Stream (fluxo) IV

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.

Saída de *bytes* – `java.io.OutputStream`

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()`

Entrada de caracteres – `java.io.Reader`

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*.

Saída de caracteres – `java.io.Writer`

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.

Persistência de dados em arquivos texto

- 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
```


Escrita/gravação em arquivos texto

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`).

Escrita sequencial em arquivos texto

`BufferredWriter`: 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.

Leitura sequencial em arquivos texto

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

```java
package usararquivos;
public class UsarArquivos {
   public static void main(String[] args) {
      String nome[] = new String[3];
      int idade[] = new int[3];
      double nota[] = new double[3];
      nome[0] = "José da Silva";
      nome[1] = "Márcia Bastos";
      nome[2] = "Carla Pereira";
      idade[0] = 23;
      idade[1] = 20;
      idade[2] = 18;
      nota[0] = 7.5;
      nota[1] = 7;
      nota[2] = 8.5;
      GerenciamentoArquivos gerente = new GerenciamentoArquivos();
      gerente.escrita("teste.dat", nome, idade, nota);
      gerente.leitura("teste.dat");
   }
}
```


Exemplo – GerenciamentoArquivos.java I

```java
package usararquivos;
import java.io.*;

public class GerenciamentoArquivos {
   public void escrita(String nomeArq, String[] vet1, int[] vet2,
    double[] vet3) {
      try {
         FileWriter arq = new FileWriter(nomeArq);
         PrintWriter out = new PrintWriter(arq);
         for (int i = 0; i < vet1.length; i++) {
            String linha = vet1[i] + ":" + vet2[i] + ":" + vet3[i];
            out.println(linha);
         }
         out.close();
      } catch (IOException erro) {
         System.out.println("Erro na escrita dos dados");
      }
   }  //fim do método escrita()

   public void leitura(String nomeArq) {
      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].
    replace(",", ".") );
30             System.out.println("Nome=" + nome + " Idade=" + idade + "
    Nota=" + nota);
31          }
32          br.close();
33       } catch (IOException erro) {
34          System.out.println("Erro na leitura dos dados");
35       }
36    } // Fim do método leitura()
37 }  // Fim da classe
```


Exemplo – GerenciamentoArquivos.java III

Persistência de dados: serialização I

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.

Persistência de dados: serialização II

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.

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

```java
package serialfarmacia;

import java.io.Serializable;

public class Medicamento implements Serializable {

   String nome;
   double preco;

   public Medicamento() {
   }

   public Medicamento(String novoNome, double novoPreco) {
      this.nome = novoNome;
      this.preco = novoPreco;
   }

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


Exemplo – Medicamento.java II

```java


   public void setPreco(double novoPreco) {
      this.preco = novoPreco;
   }

   public String getNome() {
      return this.nome;
   }

   public double getPreco() {
      return this.preco;
   }

   public void escreverMedicamento() {
      System.out.println("Nome" + this.nome);
      System.out.println("Preco" + this.preco);
   }
}
```

Exemplo – Medicamento.java III

Exemplo – TestaFarmaciaSerializacao.java I

```java
package serialfarmacia;

public class TestaFarmaciaSerializacao {

   public static void main(String[] args) {
      Farmacia ufu = new Farmacia();
      /* cadastro de medicamentos */
      Medicamento m = new Medicamento("a", 5.6);
      ufu.cadastraMedicamento(m);
      m = new Medicamento("b", 15.6);
      ufu.cadastraMedicamento(m);
      m = new Medicamento("c", 25.6);
      ufu.cadastraMedicamento(m);
      m = new Medicamento("d", 35.6);
      ufu.cadastraMedicamento(m);
      m = new Medicamento("e", 3.6);
      ufu.cadastraMedicamento(m);

      //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" +
       erro);
41         }
42      }  // Fim do método escreverMedicamentos( String )
43
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(" Preco: " + m.getPreco());
54            }
55            is.close(); arq.close();
56         } catch (IOException erro) {
57            System.out.println("Ocorreu um erro na escrita dos dados: " +
       erro);
```

Exemplo – Farmacia.java IV

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

Referências

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