

Aula 14

Arquivos, Fluxos e Serialização de Objetos

Introdução

- Dados armazenados em variáveis e arrays são temporários → são perdidos quando terminam seu escopo ou quando o programa é finalizado
- Para retenção de dados a longo prazo, os computadores utilizam **arquivos**
- Um arquivo (computacionalmente falando) é:
 - Conjunto de dados digitais que pode ser gravado em um dispositivo de armazenamento e tratado como ente único
 - Abstração de bits físicos em uma unidade lógica

Introdução

- Os computadores armazenam arquivos em **dispositivos de armazenamento secundário**: disco magnéticos, discos ópticos, fitas magnéticas, ...



<http://www.knowcomputing.com/images/STORAGE%20DEVICES.jpg>

Introdução

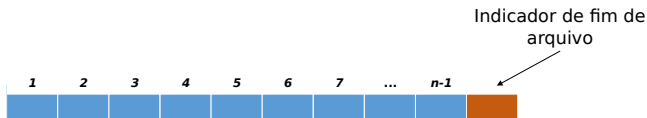
- Os dados mantidos nos arquivos são **dados persistentes**, pois existem além da duração da execução do programa
- Podemos então **recuperar esses dados na próxima execução de um programa**
- Ao lidar com arquivos, ou outras formas de ler e gravar informações, iremos utilizar o termo **“fluxo”**

Arquivos e Fluxos

- **O Java vê cada arquivo como um fluxo sequencial de bytes**
- **Cada sistema operacional fornece um mecanismo para determinar o término de um arquivo**
 - Marcador de fim de arquivo
 - Contagem do total de bytes no arquivo que é mantido na estrutura do SO
- Um programa Java que processa um fluxo de bytes simplesmente recebe uma indicação do sistema operacional quando ele alcança o fim do fluxo (independente do SO, plataforma, ...)

Arquivos e Fluxos

- Em alguns casos, a **indicação de fim de arquivo ocorre como uma exceção**
- Em outros casos, a **indicação de fim de arquivo é um valor de retorno de um método** invocado sobre um objeto que processa o fluxo



Visualização do Java de um arquivo de n bytes

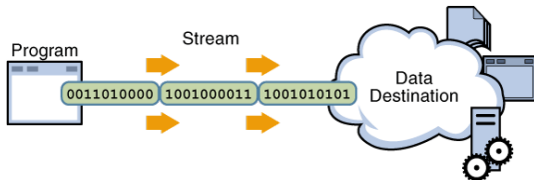
Fluxos baseados em bytes e fluxos baseados em caracteres

- Fluxos de arquivos ser utilizados para entrada e saída de dados como bytes ou caracteres
- Os fluxos de entrada e saída de *bytes* são conhecidos como **fluxos baseados em bytes**, representando os dados no seu formato binário
- Os fluxos de entrada e saída de caracteres para arquivos são conhecidos com **fluxos baseados em caracteres**, representando os dados como uma sequência de caracteres

- Arquivos criados usando fluxos baseados em *bytes* são chamados **arquivos binários** e arquivos criados usando fluxos baseados em caracteres são chamados **arquivos de texto**
- **Arquivos de texto:** podem ser lidos por editores de textos
- **Arquivos binários:** são lidos por programas que entendem o conteúdo específico do arquivo

Entrada padrão, saída padrão e fluxos de erros padrão

- Um programa Java abre um arquivo criando e associando um objeto ao fluxo de *bytes* ou de caracteres
- OBSERVAÇÃO:** o java também pode associar fluxos a diferentes dispositivos (teclado, monitor, rede, ...)



Entrada padrão, saída padrão e fluxos de erros padrão

- Por padrão, o Java cria três objetos de fluxo que são associados a dispositivos quando um programa Java inicia a execução:
 - `System.in`: objeto de fluxo de entrada padrão - normalmente permite a um programa inserir bytes a partir do teclado
 - `System.out`: objeto de fluxo de saída padrão - normalmente permite a um programa enviar a saída dos dados de caractere para a tela
 - `System.err`: objeto de fluxo de erro padrão - normalmente permite a um programa gerar a saída de mensagens de erros baseados em caractere na tela

Entrada padrão, saída padrão e fluxos de erros padrão

- Cada um dos fluxos padrão do Java podem ser redirecionados
- Ex:
 - `System.in` pode ser redirecionado para que o programa leia bytes a partir de uma origem diferente (ex: rede, arquivos, ...)
 - `System.out` e `System.err` podem ser redirecionados para uma saída diferente (ex: impressora, rede, arquivos, ...)
- A classe `System` fornece métodos `setIn`, `setOut` e `setErr` para redirecionar os fluxos de entrada, saída e erro padrão, respectivamente

O Pacote java.io

- Os programas Java realizam o processamento de arquivos utilizando classes do pacote java.io
- Esse pacote inclui definições para classes de fluxo:
 - FileInputStream: para entrada baseada em bytes de um arquivo
 - FileOutputStream: para saída baseada em bytes de um arquivo
 - FileReader: para entrada baseada em caracteres de um arquivos
 - FileWriter: para saída baseada em caracteres de um arquivo

O Pacote java.io

- **Abre-se um arquivo criando um objeto de uma dessas classes de stream**
- O construtor do objeto interage com o sistema operacional para abrir o arquivo
- **OBSERVAÇÃO:** outras classes além dessas podem ser utilizadas (<https://docs.oracle.com/javase/7/docs/api/java/io/package-tree.html>)

O Pacote java.io

- O Java também contém classes que permitem **realizar a entrada e saída de objetos ou variáveis de tipos de dados primitivos**
- Para realizar a essas entradas (gravações) e saídas (leitura) os objetos das classe `ObjectInputStream` e `ObjectOutputStream` podem ser utilizados juntos com as classes de arquivos baseados em fluxos de bytes `FileInputStream` e `FileOutputStream`

Criando um Arquivo de Texto de Acesso Sequencial

- O Java não impõe nenhuma estrutura a um arquivo
- Noções, como registros, não fazem parte da linguagem Java
- A estrutura do arquivo, portanto, é definida pelo programador

Criando um Arquivo de Texto de Acesso Sequencial

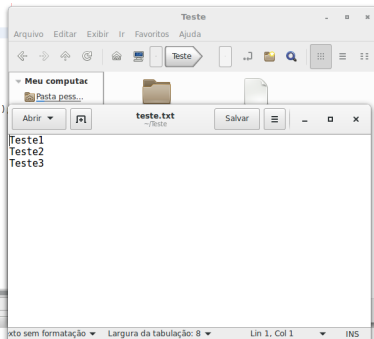
- Para **criar** ou **gravar** arquivos de texto vamos utilizar a classe `FileWriter`
- ① Basicamente, para criar um arquivo (abrir o fluxo) utiliza-se o construtor `FileWriter(caminho arquivo)`, na qual o caminho pode ser fornecido por uma `String` ou um `File`
- ② Para mandar dados via o fluxo criado, deve-se utilizar o comando `write`
- ③ Para encerrar o fluxo deve-se utilizar o comando `close`

Criando um Arquivo de Texto de Acesso Sequencial

```
8 import java.io.FileWriter;
9 import java.io.IOException;
10
11 public class Teste {
12
13     public static void main(String[] args) {
14
15         //Gravando arquivos com FileOutputStream
16         try{
17             FileWriter arquivo = new FileWriter("/home/rafael/Teste/teste.txt");
18
19             System.out.println("Gravando no arquivo...");
20
21             arquivo.write("Teste1\n");
22             arquivo.write("Teste2\n");
23             arquivo.write("Teste3\n");
24
25             arquivo.close();
26             System.out.println("Arquivo gravado com sucesso!");
27         }catch(IOException ioe){
28             System.err.println("Houve um erro ao gravar o arquivo");
29         }
30     }
31 }
32
33
```

Resultados da Pesquisa Saída - Teste (run) X

```
run:
Gravando no arquivo...
Arquivo gravado com sucesso!
```



Lendo Dados de um Arquivo de Texto de Acesso Sequencial

- Os dados são armazenados em arquivos de modo que possam ser recuperados para processamento quando necessário
- Da mesma forma que podemos gravar dados sequencialmente, podemos também ler os dados sequencialmente
- Para isso iremos utilizar a classe `FileReader`

Lendo Dados de um Arquivo de Texto de Acesso Sequencial

- Pode-se instanciar um objeto da classe `FileReader` para estabelecer o fluxo de dado com um arquivo e consequentemente realizar a leitura dos caracteres de um arquivo
- O construtor da classe `FileReader` pode receber tanto uma `String` quanto um `File` contendo o caminho do arquivo desejado

Lendo Dados de um Arquivo de Texto de Acesso Sequencial

- Utiliza-se o método `read` para ler um caractere de um arquivo
- Quando o método `read` retornar `-1` significa que o arquivo atingiu o seu final
- Utiliza-se o comando `close` para encerrar o fluxo de dado

Lendo Dados de um Arquivo de Texto de Acesso Sequencial

```
11 public class Teste {
12
13     public static void main(String[] args) {
14
15         //Gravando arquivos com FileOutputStream
16         try{
17             FileReader arquivo = new FileReader("/home/rafael/Teste/teste.txt");
18
19             System.out.println("Lendo o arquivo..");
20             char character = '.';
21             String texto = "";
22             boolean sair = false;
23             while(sair == false){
24                 int valor = arquivo.read();
25                 if(valor == -1){
26                     sair = true;
27                 }else{
28                     character = (char)valor;
29                     texto += character;
30                 }
31             }
32             arquivo.close();
33             System.out.println("Arquivo lido com sucesso!");
34             System.out.println("Imprimindo texto:");
35             System.out.println(texto);
36         }catch(IOException ioe){
37             System.err.println("Houve um erro ao gravar o arquivo");
38         }
39     }
40 }
41
42 }
```

teste.txt

teste

Salvar

teste1
teste2
teste3

sem formatação Largura da tabulação: 8 Lin 1, Col 1 INS

teste.Teste

Resultados da Pesquisa Saida x

Console do Depurador x Teste (run) x

run:
Lendo o arquivo.
Arquivo lido com sucesso!
Imprimindo texto:
teste1
teste2
teste3

Lendo Dados de um Arquivo de Texto de Acesso Sequencial

```
12 public class Teste {  
13  
14     public static void main(String[] args) {  
15  
16         //Gravando arquivos com FileOutputStream  
17         try{  
18  
19             Scanner arquivo = new Scanner(new File("/home/rafael/Teste/teste.txt"));  
20  
21             System.out.println("Lendo o arquivo..");  
22             String linha = "";  
23             while(arquivo.hasNext()){  
24                 linha = arquivo.nextLine();  
25                 System.out.println(linha);  
26             }  
27             arquivo.close();  
28             System.out.println("Arquivo lido com sucesso!");  
29         }catch(IOException ioe){  
30             System.err.println("Houve um erro ao gravar o arquivo");  
31         }  
32     }  
33  
34 }
```

Resultados da Pesquisa Salda x

Console do Depurador x Teste (run) x

run:
Lendo o arquivo..
Teste1
Teste2
Teste3
Arquivo lido com sucesso!

Atualizando Arquivos de Acesso Sequencial

- Ao abrir um arquivo existente e mandar gravar novos dados nesse arquivo da forma como foi apresentado anteriormente, a nova informação irá sobrescrever o conteúdo do arquivo existente
- Para apenas adicionar novos dados em um arquivo existente, deve abrir o arquivo em modo append
- A classe `FileWriter`, por exemplo, possui o construtor `FileWriter(String fileName, boolean append)`

Atualizando Arquivos de Acesso Sequencial

```
11 public class Teste {  
12  
13     public static void main(String[] args) {  
14  
15         //Gravando arquivos com FileOutputStream  
16         try{  
17             FileWriter arquivo = new FileWriter("/home/rafael/Teste/teste.txt", true);  
18  
19             System.out.println("Gravando novas informações no arquivo");  
20  
21             arquivo.write("Teste5\n");  
22             arquivo.write("Teste6\n");  
23             arquivo.write("Teste7\n");  
24  
25             arquivo.close();  
26             System.out.println("Arquivo gravado com sucesso!");  
27         } catch (IOException ioe){  
28             System.err.println("Erro ao gravar novas informações no arquivo");  
29         }  
30     }  
31 }  
32  
33 }
```

Resultados da Pesquisa Salda x

Console do Depurador x Teste (run) x

run:
Gravando novas informações no arquivo
Arquivo gravado com sucesso!
CONSTRUIDO COM SUCESSO (tempo total: 0 segundos)

Abrir ▾ [ícone] teste.txt ~/Teste Salvar [ícone] - □ x

Teste1
Teste2
Teste3
Teste5
Teste6
Teste7

o sem formatação ▾ Largura da tabulação: 8 ▾ Lin 1, Col 1 ▾ INS

Leitura e Escrita em *Buffer*

- Armazenamento em *buffer* (*buffering*) é uma técnica de aprimoramento do desempenho de E/S
- Nesse tipo de técnica, por exemplo, cada instrução de saída não necessariamente resulta em uma transferência física real de dados para o dispositivo de saída (uma operação lenta se comparada com as velocidades do processador e da memória principal)
- Em vez disso, cada operação de entrada/saída é dirigida para uma região na memória chamada **buffer**, que é suficientemente grande para armazenar os dados de muitas operações de entrada/saída

Leitura e Escrita em *Buffer*

- **Exemplos de classe que implementar a leitura e escrita utilizando buffer:** `BufferedReader`, `BufferedWriter`, `BufferedOutputStream`, `BufferedInputStream`, ...
- O funcionamento dessas classes é semelhante ao apresentado anteriormente
- Porém, algumas funcionalidades extras como a leitura de linhas, ao invés de caractere a caractere, são implementadas → o retorno de fim de arquivo é dada por `null`

Leitura e Escrita em *Buffer*

- **OBSERVAÇÃO 1:** utilizar o armazenamento em *buffer* pode aumentar significativamente o desempenho de um aplicativo
→ redução do número de acessos à memória
- **OBSERVAÇÃO 2:** pode-se forçar as técnicas baseadas em *buffer* a enviar os dados para os dispositivos de saída invocando o método `flush`

Leitura e Escrita em *Buffer*

```
13 public class Teste {
14     public static void main(String[] args) {
15         //Gravando arquivos com FileOutputStream
16         try{
17
18             BufferedReader arquivo = new BufferedReader(new FileReader("/home/rafael/Teste/teste.txt"));
19
20             System.out.println("Lendo o arquivo..");
21             String linha = "";
22             boolean sair = false;
23             while(sair == false){
24                 linha = arquivo.readLine();
25                 if(linha == null){
26                     sair = true;
27                 }else{
28                     System.out.println(linha);
29                 }
30             }
31             arquivo.close();
32             System.out.println("Arquivo lido com sucesso!");
33         }catch(IOException ioe){
34             System.err.println("Houve um erro ao gravar o arquivo");
35         }
36     }
37 }
38
39
40 }
```

teste.Teste

o sem formatação Largura da tabulação: 8 Lin 1. Col 1 INS

resultados da Pesquisa Saída x

Console do Depurador x Teste (run) x

run:
Lendo o arquivo..
Teste1
Teste2
Teste3
Arquivo lido com sucesso!

teste.txt
~/.teste

Salvar

Teste1
Teste2
Teste3

Fluxo de Bytes

- Vamos agora trabalhar com as classes `FileInputStream` e `FileOutputStream` para lidar com o fluxo de bytes
- A manipulação de fluxo se dá basicamente pela leitura e escrita de um array de bytes
- No caso da leitura, é retornado um valor inteiro indicando a quantidade de bytes lida → o valor -1 indica que não há mais bytes a serem lidos

Fluxo de Bytes

```
17 public class Principal {  
18  
19  
20     public static void main(String[] args) {  
21  
22         String path = "/home/rafael/Teste/batatinha_quando_nasce_sem_acento.txt";  
23  
24         try {  
25             byte[] buffer = new byte[1];  
26             FileInputStream in = new FileInputStream(path);  
27             int len = 0;  
28             while((len = in.read(buffer)) > 0){  
29                 char ch = (char)(buffer[0]);  
30                 System.out.print(ch);  
31             }  
32             in.close();  
33         } catch (IOException ex) {  
34             System.err.println("Deu pau!");  
35         }  
36     }  
37 }
```

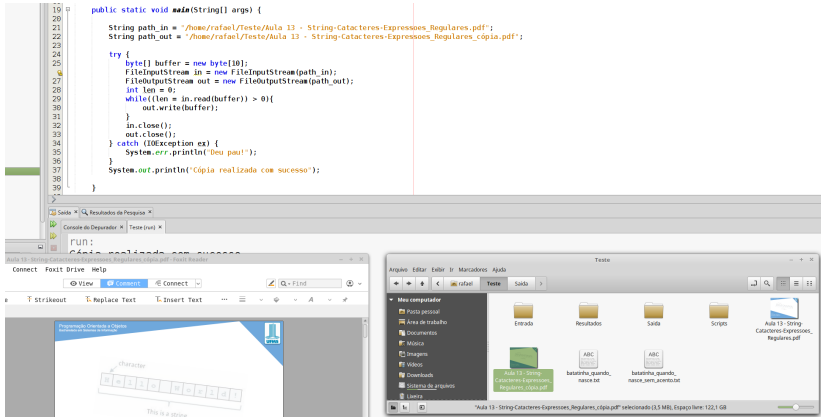
Saída x Resultados da Pesquisa x

Console do Depurador x Teste (run) x

run:

Batatinha quando nasce espalha a rama pelo chao.
Menininha quando dorme poe a mao no coracao.
Sou pequeninha do tamanho de um botao,
Carrego papai no bolso e mamae no coracao
O bolso furou e o papai caiu no chao.
Mamae que e mais querida ficou no coracao.

Fluxo de Bytes



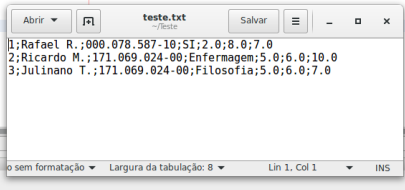
Serialização de Objetos

E se quiséssemos gravar os campos de objetos da forma como vimos até agora... como ficaria?



Serialização de Objetos

```
12 public class Teste {  
13  
14     public static void main(String[] args) {  
15  
16         //Criando um array de objetos aluno - id, nome, cpf, curso, array de notas  
17         ArrayList<Aluno> alunos = new ArrayList<Aluno>();  
18  
19         double[] notas1 = {2,8,7};  
20         alunos.add(new Aluno(1, "Rafael R.", "000.078.587-10", "SI", notas1));  
21         double[] notas2 = {5,6,10};  
22         alunos.add(new Aluno(2, "Ricardo M.", "171.069.024-00", "Enfermagem", notas2));  
23         double[] notas3 = {5,6,7};  
24         alunos.add(new Aluno(3, "Juliano T.", "171.069.024-00", "Filosofia", notas3));  
25  
26         try{  
27             System.out.println("Gravando arquivo");  
28             FileWriter arq = new FileWriter("/home/rafael/Teste/teste.txt");  
29             for (Aluno aluno : alunos){  
30                 arq.write(aluno.toString() + "\n");  
31             }  
32             arq.close();  
33             System.out.println("Arquivo Gravado com Sucesso!");  
34         } catch (IOException ioe){  
35             System.err.println("Erro ao gravar arquivo");  
36         }  
37     }  
38  
39 }  
40
```



Abrir [ícone] teste.txt [ícone] Salvar [ícone] [ícone] [ícone] [ícone]

```
1;Rafael R.;000.078.587-10;SI;2.0;8.0;7.0  
2;Ricardo M.;171.069.024-00;Enfermagem;5.0;6.0;10.0  
3;Juliano T.;171.069.024-00;Filosofia;5.0;6.0;7.0
```

sem formatação ▾ Largura da tabulação: 8 ▾ Lin 1. Col 1 ▾ INS

teste.Teste > main > try >

Resultados da Pesquisa Saída x

Console do Depurador x Teste (run) x

```
run:  
Gravando arquivo  
Arquivo Gravado com Sucesso!
```

Serialização de Objetos

```
14 public static void main(String[] args) {
15
16     //Tentativa de recriar os objetos a partir dos dados gravados no arquivo
17     ArrayList<Aluno> alunos = new ArrayList<Aluno>();
18     try{
19         BufferedReader arq = new BufferedReader(new FileReader("/home/rafael/Teste/teste.txt"));
20
21         String linha = "";
22         while( (linha = arq.readLine()) != null ){
23             String[] partes = linha.split(";");
24             int id = Integer.parseInt(partes[0]);
25             double[] notas = new double[3];
26             for(int i=0; i<notas.length; i++){
27                 notas[i] = Double.parseDouble(partes[4+i]);
28             }
29             alunos.add(new Aluno(id,partes[1],partes[2],partes[3],notas));
30         }
31     }
32     arq.close();
33     catch(Exception e){
34         System.err.println("Erro ao ler o arquivo");
35     }
36
37     //Imprimindo o array de Alunos
38     for(Aluno aluno : alunos){
39         System.out.println(aluno.toString());
40     }
41 }
42 }
```

Abrir

teste.txt

Salvar

Teste

1;Rafael R.;000.078.587-10;SI;2.0;8.0;7.0

2;Ricardo M.;171.069.024-00;Enfermagem;5.0;6.0;10.0

3;Juliano T.;171.069.024-00;Filosofia;5.0;6.0;7.0

Resultados da Pesquisa

Salida

sem formatação

Largura da tabulação: 8

Lin 1, Col 1

INS

Console do Depurador

Teste (run)

run:

1;Rafael R.;000.078.587-10;SI;2.0;8.0;7.0

2;Ricardo M.;171.069.024-00;Enfermagem;5.0;6.0;10.0

3;Juliano T.;171.069.024-00;Filosofia;5.0;6.0;7.0

Serialização de Objetos

- No exemplo anterior, foram gravados e lidos todos os atributos/campos de um objetos
- Com base na seção de Hierarquia de Dados apresentada anteriormente, podemos considerar cada atributo de um objeto como sendo um campo e cada objeto como sendo um registro
- Considerando isso, pode-se tornar interessante gravar um objeto e, posteriormente, ler um objeto
- Para gravar e ler objetos, o Java fornece um mecanismo de chamado de **serialização de objetos**

Serialização de Objetos

- Um **objeto serializado** é um objeto representado como uma sequência de *bytes* que inclui os dados do objeto bem como as informações sobre o tipo do objeto e os tipos dos dados armazenados no objeto
- Para serializar um objeto, e consequentemente possibilitar sua gravação e leitura, é necessário implementar a interface `Serializable` na classe do objeto que se deseja ler/gravar
- A interface `Serializable` não possui métodos ou campos e **serve apenas para identificar a semântica de ser serializado**

Serialização de Objetos

- Tentar gravar um objeto cuja classe não implementa a interface `Serializable` gerará uma exceção (`NotSerializableException`)
- Depois que um objeto serializado foi gravado em um arquivo, ele pode ser lido a partir do arquivo e **desserializado**, isto é, **as informações dos tipos e os bytes que representam o objeto e seus dados podem ser utilizadas para recriar o objeto na memória**

Serialização de Objetos

```
9
10
11 public class Aluno implements Serializable {
12
13     private int id;
14     private String nome;
15     private String CPF;
16     private String curso;
17     private double[] notas;
18
19     public Aluno(int id, String nome, String CPF, String curso, double[] notas) {
20         this.id = id;
21         this.nome = nome;
22         this.CPF = CPF;
23         this.curso = curso;
24         this.notas = notas;
25     }
26
27     public int getId() {
28         return id;
29     }
30
31     public void setId(int id) {
32         this.id = id;
33     }
34 }
```

Serialização de Objetos

- **OBSERVAÇÃO 1:** caso queira que um campo de um objeto não seja serializado, basta utilizar a palavra-chave `transient` antes do tipo do campo
- **OBSERVAÇÃO 2:** campos não serializados são inicializados com os valores padrão

Classes `ObjectInputStream` e `ObjectOutputStream`

- As classe `ObjectInputStream` e `ObjectOutputStream` implementam as interfaces `ObjectInput` e `ObjectOutput`, respectivamente
- Essas classes permitem que objetos inteiros sejam lidos ou gravados em fluxo (possivelmente um arquivo)

Classes `ObjectInputStream` e `ObjectOutputStream`

- Esses objetos devem ser inicializados com classes que leem e gravam arquivos em fluxo → `FileInputStream` e `FileOutputStream`

```
ObjectInputStream inpStream = new ObjectInputStream(new FileInputStream("texto5.abc"));
```

```
ObjectOutputStream outStream = new ObjectOutputStream(new FileOutputStream("texto5.abc"));
```

Interfaces `ObjectOutput` e `ObjectInput`

- A interface `ObjectOutput` contém o método `writeObject`, que recebe um `Object` como um argumento e grava suas informações em um `OutputStream`
- Uma classe que implementa a interface `ObjectOutput` declara esse método e assegura que o objeto sendo gerado implementa a interface `Serializable`

Interfaces `ObjectOutput` e `ObjectInput`

- De maneira semelhante, a interface `ObjectInput` contém o método `readObject`, que lê e retorna uma referência a um `Object` a partir de um `InputStream`
- Depois que um objeto foi lido, podemos fazer uma coerção (cast) da sua referência para o tipo real do objeto

Criando arquivos de acesso sequencial com a serialização de objetos

```
1 import java.io.FileOutputStream;
2 import java.io.IOException;
3 import java.io.ObjectOutputStream;
4 import java.util.ArrayList;
5
6 public class Teste {
7
8     public static void main(String args[]){
9
10         ArrayList<Aluno> alunos = new ArrayList<Aluno>();
11         alunos.add(new Aluno("Rafael R.", "000.000.000-00", "12314564", "Sistemas de Informação"));
12         alunos.add(new Aluno("Ricardo M.", "111.111.111-11", "5789745", "Enfermagem"));
13         alunos.add(new Aluno("Ronaldo F.", "222.222.222-22", "78975456", "Medicina"));
14
15         ObjectOutputStream outStream = null;
16         try {
17             outStream = new ObjectOutputStream(new FileOutputStream("dados.dat"));
18             for(Aluno al : alunos){
19                 outStream.writeObject(al);
20             }
21             System.out.println("Informações adicionadas no arquivo com sucesso");
22         } catch (IOException ex) {
23             ex.printStackTrace();
24         } finally{
25             try {
26                 outStream.close();
27             } catch (IOException ex) {
28                 ex.printStackTrace();
29             }
30         }
31     }
32 }
33
34
```

Localizar: System.out.print

Saida x

Console do Depurador x Teste (run) x

run:
Informações adicionadas no arquivo com sucesso

Lendo arquivos de acesso sequencial com a desserialização de objetos

```
7 public class Teste {  
8  
9     public static void main(String args[]){  
10  
11         ArrayList<Aluno> alunos = new ArrayList<Aluno>();  
12  
13         ObjectInputStream inStream = null;  
14         try {  
15             inStream = new ObjectInputStream(new FileInputStream("dados.dat"));  
16  
17             while(true){  
18                 Aluno aluno = (Aluno)inStream.readObject();  
19                 alunos.add(aluno);  
20             }  
21  
22         }catch (EOFException ex){  
23             //fim do arquivo foi alcançado  
24  
25         }catch (ClassNotFoundException ex){  
26             ex.printStackTrace();  
27         }  
28         catch (IOException ex) {  
29             ex.printStackTrace();  
30         }  
31         finally{  
32             try {  
33                 inStream.close();  
34             } catch (IOException ex) {  
35                 ex.printStackTrace();  
36             }  
37         }  
38  
39         for(Aluno aluno : alunos){  
40             System.out.println(aluno.toString());  
41         }  
42     }  
43 }
```

Localizar: System.out.print

Anterior Próximo

Saída x

Console do Depurador x Teste (run) x

run:
Nome: Rafael R. CPF: 000.000.000-00 RGA: 12314564 Curso: Sistemas de Informação
Nome: Ricardo W. CPF: 111.111.111-11 RGA: 5789745 Curso: Enfermagem
Nome: Ronaldo F. CPF: 222.222.222-22 RGA: 78975456 Curso: Moda

Lendo arquivos de acesso sequencial com a desserialização de objetos

- **OBSERVAÇÃO 1:** o método `readObject` lança uma `EOFException` se ocorrer uma tentativa de leitura depois do fim do arquivo
- **OBSERVAÇÃO 2:** o método `readObject` lança uma `ClassNotFoundException` se a classe para o objeto sendo lido não puder ser localizada
- **OBSERVAÇÃO 3:** se você serializar um objeto e depois modificar a classe do objeto, não conseguirá mais desserializá-lo → `InvalidClassException`

Lendo arquivos de acesso sequencial com a desserialização de objetos

Legal... mas um `ArrayList` também não é um objeto??



Lendo arquivos de acesso sequencial com a desserialização de objetos

```
1 import java.io.FileOutputStream;
2 import java.io.IOException;
3 import java.io.ObjectOutputStream;
4 import java.util.ArrayList;
5
6 public class Teste {
7
8     public static void main(String args[]){
9
10         ArrayList<Pessoa> pessoas = new ArrayList<Pessoa>();
11
12         pessoas.add(new Pessoa("Rafael", "234654"));
13         pessoas.add(new Pessoa("Ricardo", "789789"));
14         pessoas.add(new Pessoa("Ronaldo", "1712469"));
15
16         try {
17             ObjectOutputStream arq = new ObjectOutputStream(new FileOutputStream("/home/rafael/Teste/Entrada/Classel/teste5.reg"));
18             arq.writeObject(pessoas);
19             arq.close();
20             System.out.println("Arquivo gravado com sucesso");
21         } catch (IOException ex) {
22             ex.printStackTrace();
23         }
24     }
25 }
26
27 }
```


Lendo arquivos de acesso sequencial com a desserialização de objetos

```
1 import java.io.FileInputStream;
2 import java.io.IOException;
3 import java.io.ObjectInputStream;
4 import java.util.ArrayList;
5
6 public class Teste {
7
8     public static void main(String args[]){
9
10         try {
11             ObjectInputStream arq = new ObjectInputStream(new FileInputStream("/home/rafael/Teste/Entrada/Classe1/teste5.reg"));
12
13             try {
14                 ArrayList<Pessoa> pessoas = (ArrayList<Pessoa>)arq.readObject();
15                 for(Pessoa p: pessoas){
16                     System.out.println(p.toString());
17                 }
18             } catch (ClassNotFoundException ex) {
19                 ex.printStackTrace();
20             }
21
22             } catch (IOException ex) {
23                 ex.printStackTrace();
24             }
25         }
26     }
27 }
28
29
```

Salida X | Matcher.java X | TCT.java X | Main.java X | ObjectStreamClass.java X

Console do Depurador X | Teste (run) X

run:
Nome: Rafael CPF: 234654
Nome: Ricardo CPF: 789789
Nome: Ronaldo CPF: 1712469

Gravando Tipos Primitivos e Tipos por Referência

Gravando Tipos Primitivos e Tipos por Referência

```
15 public class Teste2 {  
16  
17     public static void main(String[] args){  
18  
19         try{  
20             ObjectInputStream arqObj = new ObjectInputStream(new FileInputStream("/home/rafael/Teste/teste.obj"));  
21  
22             String nome = (String)arqObj.readObject();  
23             int idade = arqObj.readInt();  
24             double salario = arqObj.readDouble();  
25  
26             System.out.println("Nome: " + nome);  
27             System.out.println("Idade: " + idade);  
28             System.out.println("Salario: " + salario);  
29         }catch(Exception e){  
30             System.err.println("Deu pau!");  
31         }  
32     }  
33 }  
34  
35 }
```

Resultados da Pesquisa

Saída - Teste (run) x

```
run:  
Nome: Rafael  
Idade: 10  
Salario: 900.0
```

Classe File

- Frequentemente quando vamos trabalhar com fluxos em arquivos, temos que verificar se um arquivo existe, listar arquivos de um diretório para abrir todos, criar diretórios, etc;;
- **A classe File é útil para recuperar informações sobre arquivos ou diretórios em disco**
- Os objetos da classe File não abrem arquivos nem fornecem quaisquer capacidades de processamento de arquivos
- Entretanto, os objetos File são utilizados frequentemente como objetos de outras classes `java.io` para especificar arquivos ou diretórios a manipular

Criando objetos File

- A classe File fornece 4 construtores:
 - `File(String pathname)`: cria um novo objeto File convertendo a *string* em um caminho de arquivo
 - `File(File parent, String child)`: cria um novo objeto File de um arquivo de um File pai (*parent*) acrescentado de uma *string* contendo um caminho filho (*child*)
 - `File(String parent, String child)`: cria um novo objeto File de um arquivo de um File pai (*parent*) acrescentado de uma *string* contendo um caminho filho (*child*)
 - `File(URI uri)`: cria um novo objeto File convertendo uma URI em um caminho

Criando objetos File

- Um caminho de arquivo ou diretório especifica sua localização em disco
- O caminho inclui alguns ou os principais diretórios para o arquivo ou diretório
- Um **caminho absoluto** contém todos os diretórios, desde o diretório-raiz, que levam a um arquivo ou diretório específico
- Um **caminho relativo** normalmente inicia a partir do diretório no qual o aplicativo começou a executar e, portanto, é “relativo” ao diretório atual

Criando objetos File

- Criando um File com caminho absoluto

```
File file = new File("/home/rafael/Teste/Entrada/Classel/texto1.txt");
```

- Criando um File com caminho relativo (mesmo diretório onde o programa está rodando)

```
File file = new File("texto1.txt");
```

- Criando um File com caminho relativo (mesmo diretório onde o programa está rodando)

```
File file = new File("./texto1.txt");
```

- Criando um File com caminho relativo (dois diretórios acima de onde o programa está rodando)

```
File file = new File("../..//texto1");
```

Alguns Métodos

- Alguns métodos interessantes da classe File são:
 - **exists**: retorna true se o caminho ou o diretório existe
 - **isDirectory**: retorna true se o caminho do objeto File é um diretório
 - **isFile**: retorna true se o caminho do objeto File é um arquivo
 - **getAbsolutePath**: retorna o caminho absoluto do objeto File
 - **mkdir**: cria um diretório

Alguns Métodos

- Alguns métodos interessantes da classe File são:
 - **getName**: retorna somente o nome do arquivo
 - **lastModified**: retorna a data da última modificação do arquivo
 - **length**: retorna o tamanho do arquivo em bytes
 - **getParent**: retorna o diretório pai de um arquivo

Demonstrando a classe File

```
11 public class Teste {  
12  
13     public static void main(String[] args) {  
14  
15         File file = new File("/home/rafael/Área de trabalho/UFMS/Disciplinas/P00_1_2016/"  
16             + "Aulas/Aula 14 - Arquivos, Fluxos e Serialização de Objetos/figuras");  
17  
18         System.out.println("O arquivo/diretório existe? " + file.exists());  
19         System.out.println("É um diretório? " + file.isDirectory());  
20         System.out.println("É um arquivo? " + file.isFile());  
21         System.out.println("Caminho da variável file: " + file.getAbsolutePath());  
22         System.out.println("Data da última modificação: " + new Date(file.lastModified()).toString());  
23  
24         System.out.println("Listando arquivos dentro do diretório:");  
25         File[] files = file.listFiles();  
26         for(int f=0;f<files.length;f++){  
27             System.out.println("- " + files[f].getAbsolutePath());  
28         }  
29     }  
30 }  
31  
32
```

Resultados da Pesquisa Salda - Teste (run) x

```
run:  
O arquivo/diretório existe? true  
É um diretório? true  
É um arquivo? false  
Caminho da variável file: /home/rafael/Área de trabalho/UFMS/Disciplinas/P00_1_2016/Aulas/Aula 14 - Arquivos, Fluxos e Serialização de Objetos/figuras  
Data da última modificação: Wed Jul 13 13:55:51 AMT 2016  
Listando arquivos dentro do diretório:  
- /home/rafael/Área de trabalho/UFMS/Disciplinas/P00_1_2016/Aulas/Aula 14 - Arquivos, Fluxos e Serialização de Objetos/figuras/capa.pdf  
- /home/rafael/Área de trabalho/UFMS/Disciplinas/P00_1_2016/Aulas/Aula 14 - Arquivos, Fluxos e Serialização de Objetos/figuras/ilustracao_hierarquia_dados.png  
- /home/rafael/Área de trabalho/UFMS/Disciplinas/P00_1_2016/Aulas/Aula 14 - Arquivos, Fluxos e Serialização de Objetos/figuras/principais_metodos_file.png  
- /home/rafael/Área de trabalho/UFMS/Disciplinas/P00_1_2016/Aulas/Aula 14 - Arquivos, Fluxos e Serialização de Objetos/figuras/ilustracao_fluxo_arquivos.png
```

Demonstrando a classe File

```
11 public class Teste {  
12  
13     public static void main(String[] args) {  
14  
15         File file = new File("/home/rafael/experimentos_novos/Representacoes/20ng_sparse.arff");  
16  
17         System.out.println("O arquivo/diretório existe? " + file.exists());  
18         System.out.println("É um diretório? " + file.isDirectory());  
19         System.out.println("É um arquivo? " + file.isFile());  
20         System.out.println("Caminho da variável completo de file: " + file.getAbsolutePath());  
21         System.out.println("Somente o nome do arquivo: " + file.getName());  
22         System.out.println("Data da última modificação: " + new Date(file.lastModified()).toString());  
23         System.out.println("Tamanho do arquivo (em bytes): " + file.length());  
24         System.out.println("Diretório pai: " + file.getParent());  
25     }  
26  
27 }  
28
```

Resultados da Pesquisa

Saída - Teste (run) x

```
run:  
O arquivo/diretório existe? true  
É um diretório? false  
É um arquivo? true  
Caminho da variável completo de file: /home/rafael/experimentos_novos/Representacoes/20ng_sparse.arff  
Somente o nome do arquivo: 20ng_sparse.arff  
Data da última modificação: Fri Apr 15 14:07:44 AMT 2016  
Tamanho do arquivo (em bytes): 12801867  
Diretório pai: /home/rafael/experimentos_novos/Representacoes
```

Demonstrando a classe File

- **OBSERVAÇÃO 1:** um caractere separador é utilizado para separar diretórios e arquivo no caminho. No Windows, o caractere separador é uma barra invertida (\). Em Unix, o caractere separador é uma (/)
- **OBSERVAÇÃO 2:** O java processa os caracteres de separação do Windows e do UNIX de maneira idêntica
Ex: `c:\Program Files\Java\jdk11.8.0_11\bin`

Demonstrando a classe File

- **OBSERVAÇÃO 3:** o campo estático `File.separator` retorna o separador do SO local
- **OBSERVAÇÃO 4:** Utilizar `\` como separador de diretório em vez de `\\` em uma literal de *string* é um erro de lógica, uma vez que uma `\` simples indica que o próximo caractere representará uma sequência de escape

Classe Files

- O Java também possui a classe Files, a qual contém uma série de métodos estáticos, para manipular e extrair informações do sistema de arquivos, mas manipular os próprios arquivos
- Porém, com a manipulação apenas por chamadas de métodos estáticos, a classe Files irá remeter a uma programação procedural
- Documentação: <https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html>

Classe Path

- Vários dos métodos da classe Files recebem como parâmetro objetos da interface Path
- A interface Path contém uma série de métodos para a manipulação do sistema de arquivos assim como a classe File
- Documentação: <https://docs.oracle.com/javase/8/docs/api/java/nio/file/Path.html>
- Uma das formas de criar um Path é utilizando a classe Paths

Classe Path

```
15 public class Principal {  
16  
17     public static void main(String[] args) {  
18         Path path = Paths.get("/home/rafael/Downloads");  
19  
20         System.out.println(path);  
21         System.out.println("Obtendo o sistema de arquivos: " + path.getFileSystem());  
22         System.out.println("Obtendo a pasta raiz: " + path.getRoot());  
23         System.out.println("Obtendo a URI do path: " + path.toUri());  
24     }  
25 }  
26
```

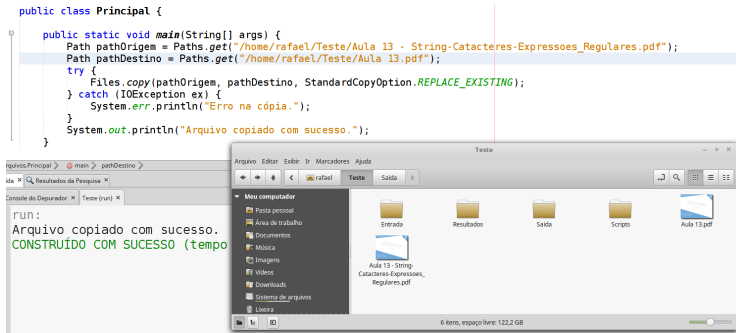
Saída x Resultados da Pesquisa x

Console do Depurador x TextCategorizationTool_MultiThreading (run) x TextCategorizationTool_MultiThreading (run) #2 x Teste (run) x

run:
/home/rafael/Downloads
Obtendo o sistema de arquivos: sun.nio.fs.LinuxFileSystem@15db9742
Obtendo a pasta raiz: /
Obtendo a URI do path: file:///home/rafael/Downloads/

Classe Files

- Copiando arquivos:

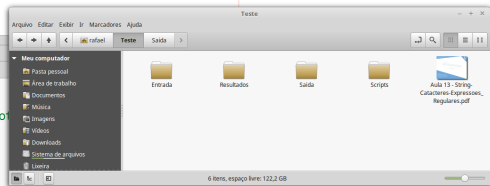


Classe Files

Deletando arquivos:

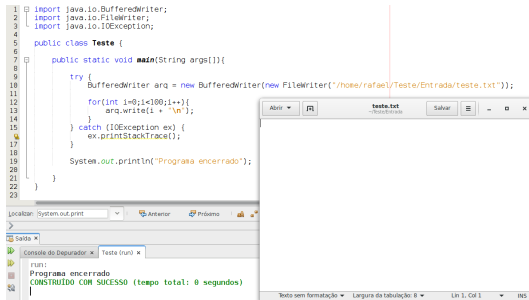
```
20 public class Principal {  
21  
22     public static void main(String[] args) {  
23         Path pathOrigem = Paths.get("/home/rafael/Teste/Aula 13 - String-Catacteres-Expressoes_Regulares.pdf");  
24         Path pathDestino = Paths.get("/home/rafael/Teste/Aula 13.pdf");  
25  
26         try {  
27             Files.deleteIfExists(pathDestino);  
28         } catch (IOException ex) {  
29             System.err.println("Erro ao apagar um arquivo");  
30         }  
31         System.out.println("Arquivo deletado com sucesso");  
32     }  
33 }  
34  
35
```

run:
Arquivo deletado com sucesso
CONSTRUIDO COM SUCESSO (tempo total: 0s)



Importância de se fechar arquivos

- Fechar um arquivo (`.close()`) é importante não só por questões envolvendo a tabela de descritores de arquivos de um sistema operacional mas também para gravar de fato as informações em um arquivo



```
1 import java.io.BufferedWriter;
2 import java.io.FileWriter;
3 import java.io.IOException;
4
5 public class Teste {
6
7     public static void main(String args[]){
8
9         try {
10             BufferedWriter arq = new BufferedWriter(new FileWriter("/home/rafael/Teste/Entrada/teste.txt"));
11
12             for(int i=0;i<100;i++){
13                 arq.write(i + "\n");
14             }
15         } catch (IOException ex) {
16             ex.printStackTrace();
17         }
18
19         System.out.println("Programa encerrado");
20     }
21 }
22
23
```

Localizar: System.out.print Anterior Próximo

Salida

Console do Depurador x Teste (run) x

run:
Programa encerrado
CONSTRUIDO COM SUCESSO (tempo total: 0 segundos)

teste.txt
-Rafael

Salvar

Texto sem formatação Largura da tabulação: 8 Lin 1, Col 1 INS

Importância de se fechar arquivos

- Agora como fica utilizando o método close()

The screenshot shows an IDE with a Java file named `Teste.java`. The code imports `java.io.BufferedWriter`, `java.io.FileWriter`, and `java.io.IOException`. It defines a `Teste` class with a `main` method. Inside `main`, a `BufferedWriter` object is created, and a loop writes numbers 0 to 100 to a file named `teste.txt`. After the loop, `arq.close();` is called. A `catch` block handles `IOException`. The program prints "Programa encerrado". The IDE's console shows the output: "Programa encerrado" and "CONSTRUIDO COM SUCESSO (tempo total: 0 segundos)". To the right, a preview of `teste.txt` shows the numbers 0 through 10.

```
1 import java.io.BufferedWriter;
2 import java.io.FileWriter;
3 import java.io.IOException;
4
5 public class Teste {
6
7     public static void main(String args[]){
8
9         try {
10             BufferedWriter arq = new BufferedWriter(new FileWriter("/home/rafael/Teste/Entrada/teste.txt"));
11
12             for(int i=0;i<100;i++){
13                 arq.write(i + "\n");
14             }
15
16             arq.close();
17         } catch (IOException ex) {
18             ex.printStackTrace();
19         }
20
21         System.out.println("Programa encerrado");
22     }
23 }
```

Localizar: System.out.print | Anterior | Próximo

Teste > main > try >

Salida x

Console do Depurador x Teste (run) x

run:
Programa encerrado
CONSTRUIDO COM SUCESSO (tempo total: 0 segundos)

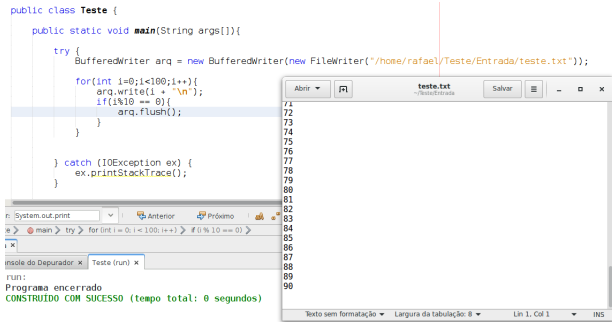
teste.txt
~TesteEntrada

0
1
2
3
4
5
6
7
8
9
10

Texto sem formatação | Largura da tabulação: 8 | Lin 1, Col 1 | INS

Método flush

- Pode-se utilizar o método `flush()` para forçar que os dados sejam gravados em um arquivo antes do programa chegar no método `close()`



Método flush

- Vale ressaltar que algumas classes que lidam com envio de dados em fluxo implementam um conceito denominado **auto-flush**, isto é, a cada envio de dados o flush é realizado automaticamente
- Por exemplo, quando redirecionamos o fluxo do `System.out` para um arquivo, cada comando `println` já realizava uma gravação no arquivo

RandomAccessFile

- Até o momento, vimos métodos de leitura sequencial, na qual o arquivo pode ser visto como um array e cada operação de leitura e escrita move o ponteiro do arquivo para a próxima posição do array
- Porém, se houver situações em que desejamos voltar ou avançar com o ponteiro do arquivo para fazer uma leitura ou escrita podemos usar a classe `RandomAccessFile`
- Essa classe também possui métodos do tipo `read` e `write`, além do método `seek` que permite especificar a posição do ponteiro do arquivo

RandomAccessFile

```
1 import java.io.IOException;
2 import java.io.RandomAccessFile;
3 import java.util.ArrayList;
4
5 public class Teste {
6
7     public static void main(String args[]){
8
9         try {
10             RandomAccessFile teste = new RandomAccessFile("/home/rafael/Teste/Entrada/Classel/teste5.txt", "rw");
11
12             teste.writeUTF("Podendo usar o RandomAccessFile...");
13             teste.seek(1);
14             teste.writeChar('F');
15             teste.close();
16
17
18             teste = new RandomAccessFile("/home/rafael/Teste/Entrada/Classel/teste5.txt", "rw");
19             String linha = "";
20             while( (linha = teste.readLine()) != null){
21                 System.out.println(linha);
22             }
23
24         } catch (IOException ex) {
25             ex.printStackTrace();
26         }
27     }
28 }
29
30
31
```

Saldas x Matcher.java x TCT.java x Main.java x ObjectOutputStreamClass.java x

Console do Depurador x Teste (run) x

run:
Podendo usar o RandomAccessFile...

Codificações de Caracteres

- Pode-se utilizar diferentes esquemas de codificações para representar os caracteres
- Esses esquemas diferem entre si pela quantidade de caracteres que podem armazenar e os tipos de caracteres que podem armazenar
- Os esquema de codificação (Charsets) padrões são: US-ASCII, ISO-8859-1, UTF-8 e UTF-16

Codificações de Caracteres

- Se quiser ler ou gravar especificando o esquema de codificação, pode-se utilizar a classe `InputStreamReader` e `OutputStreamReader` respectivamente

```
BufferedReader txtFile = new BufferedReader(new InputStreamReader(new FileInputStream(file), charset));
```

Abrindo Arquivos com JFileChooser

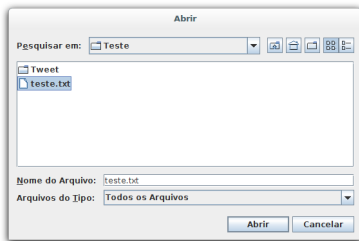
- A classe JFileChooser exibe uma caixa de diálogo que permite ao usuário selecionar facilmente arquivos ou diretórios
- Pode ser usado tanto para definir caminhos para abrir arquivos e diretórios quanto para salvar arquivos
- Quem define isso são os métodos `showOpenDialog` e `showSaveDialog` respectivamente

Abrindo Arquivos com JFileChooser

- Métodos interessantes:
 - `setFileSelectionMode(int tipo)`: define se pode ser selecionados somente arquivos, diretórios ou arquivos e diretórios
 - `showOpenDialog(Component componente)`: exibe uma caixa de diálogo para o usuário selecionar um arquivo/diretório
 - `showSaveDialog(Component componente)`: exibe uma caixa de diálogo para que o usuário possa definir um diretório ou caminho completo para salvar um arquivo
 - `getSelectedFile()`: obtém o arquivo/diretório selecionado pelo usuário
- **OBSERVAÇÃO:** a classe `JFilechooser` possui uma classe de valores pré-definidos para facilitar a programação

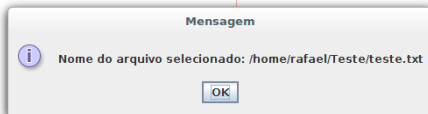
Abrindo Arquivos com JFileChooser

```
6 package teste;
7
8 import java.io.File;
9 import javax.swing.JFileChooser;
10 import javax.swing.JOptionPane;
11
12 public class Teste {
13
14     public static void main(String[] args) {
15
16         JFileChooser fileChooser = new JFileChooser();
17         fileChooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
18
19         int retorno = fileChooser.showOpenDialog(null);
20         if(retorno == JFileChooser.CANCEL_OPTION){
21             JOptionPane.showMessageDialog(null, "O usuário clicou no CANCELAR");
22         }
23
24         File file = fileChooser.getSelectedFile();
25         if(file == null || file.getName().equals("")){
26             JOptionPane.showMessageDialog(null, "Nome de arquivo Inválido");
27         }
28
29         JOptionPane.showMessageDialog(null, "Nome do arquivo selecionado: " + file.getAbsolutePath());
30     }
31 }
32
33 }
```



Abrindo Arquivos com JFileChooser

```
6 package teste;
7
8 import java.io.File;
9 import javax.swing.JFileChooser;
10 import javax.swing.JOptionPane;
11
12 public class Teste {
13
14     public static void main(String[] args) {
15
16         JFileChooser fileChooser = new JFileChooser();
17         fileChooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
18
19         int retorno = fileChooser.showOpenDialog(null);
20         if(retorno == JFileChooser.CANCEL_OPTION){
21             JOptionPane.showMessageDialog(null, "O usuário clicou no CANCELAR");
22             return;
23         }
24
25         File file = fileChooser.getSelectedFile();
26         if(file == null || file.getName().equals("")) {
27             JOptionPane.showMessageDialog(null, "Nome de arquivo Inválido");
28             return;
29         }
30
31         JOptionPane.showMessageDialog(null, "Nome do arquivo selecionado: " + file.getAbsolutePath());
32     }
33 }
34
35 }
```



Abrindo Arquivos com JFileChooser (com filtro por mais de tipo de arquivo)

```
JFileChooser load = new JFileChooser();  
load.setFileSelectionMode(load.FILES_ONLY);  
load.setDialogTitle("Selecionar Imagem");  
load.setDialogType(load.OPEN_DIALOG);  
load.addChoosableFileFilter(new FileNameExtensionFilter("Portable Network Graphics (*.png)", "png"));  
load.addChoosableFileFilter(new FileNameExtensionFilter("Joint Photographics Experts Group (*.jpg)", "jpg", "jpeg"));  
load.showSaveDialog(null);  
  
File config = load.getSelectedFile();  
if(config == null){  
    return;  
}
```

Exercício Conta Bancária

- Complementar o Projeto Banco
 - Salvar dados ao fechar o programa
 - Carregar dados ao carregar o programa
 - Criar uma classe com métodos estáticos para tal finalidade

Material Complementar

- Capítulo 15 - Pacote java.io

<https://www.caelum.com.br/apostila-java-orientacao-objetos/pacote-java-io/>

- Leitura e escrita de arquivos de texto em Java

<http://www.devmedia.com.br/leitura-e-escrita-de-arquivos-de-texto-em-java/25529>

Material Complementar

- Class Charset

`https:`

`//docs.oracle.com/javase/7/docs/api/java/nio/charset/Charset.html`

- Charsets e encodings

`http://revolucao.etc.br/archives/charsets-e-encodings/`

- Lesson: Basic I/O

`https://docs.oracle.com/javase/tutorial/essential/io/index.html`

Imagem do Dia



Programação Orientada a Objetos

<http://lives.ufms.br/moodle/>

Rafael Geraldeli Rossi
rafael.g.rossi@ufms.br

Slides baseados em [Deitel and Deitel, 2010]

Referências Bibliográficas I



Deitel, P. and Deitel, H. (2010).

Java: How to Program.

How to program series. Pearson Prentice Hall, 8th edition.