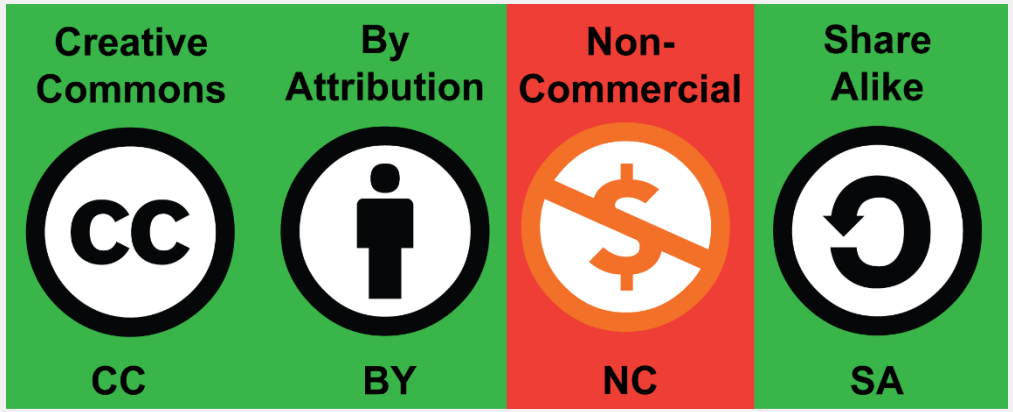
Manipulação de Arquivos em Java

Prof. MSc. Jackson Antonio do Prado Lima
jacksonpradolima at gmail.com

Departamento de Sistemas de Informação – DSI

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Licença



UDESC

Histórico de Modificação

- Esta apresentação possui contribuição dos seguintes professores:
  - Fernando José Muchalski
  - Alex Luiz de Souza
  - Anderson Fabiano Dums
  - Jackson Antonio do Prado Lima

UDESC

Agenda

- Introdução
- Tipos de Fluxo de Dados e de Arquivos
- Manipulação de Arquivos em Java
  - Classe *File*
  - Classes *FileReader* e *FileWriter*
  - Classes *FileInputStream* e *FileOutputStream*
  - Serialização de Objetos
  - Atividade de Laboratório

UDESC

Introdução

- Para que um dado possa ser manipulado ele deve estar na memória volátil (memória temporária) do computador.
  - Variáveis declaradas de um programa.
  - Estes dados são perdidos quando o programa é finalizado.

Introdução

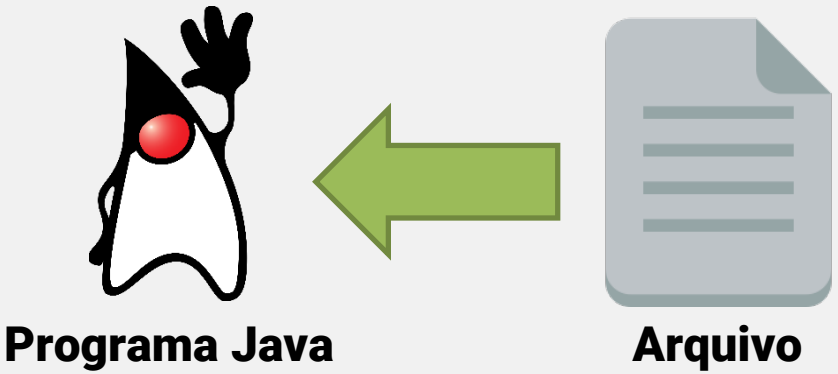
- O que fazer para que os dados não sejam perdidos?
  - Armazená-los em memória não volátil, por exemplo, na forma de arquivos e trazê-los para a memória não volátil quando necessário manipulá-los.

Fluxo de Entrada e Saída

Esta transferência de dados entre a aplicação e o arquivo pode ser de entrada ou de saída.

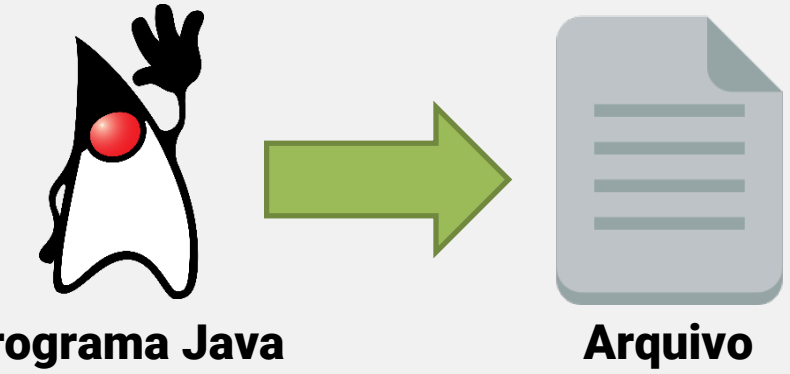
Fluxo de Entrada (*InputStream*)

- Quando o aplicação lê os dados de um arquivo e os armazena em uma **variável**.



Fluxo de Saída (*OutputStream*)

- A aplicação salva o valor de uma variável em um arquivo.



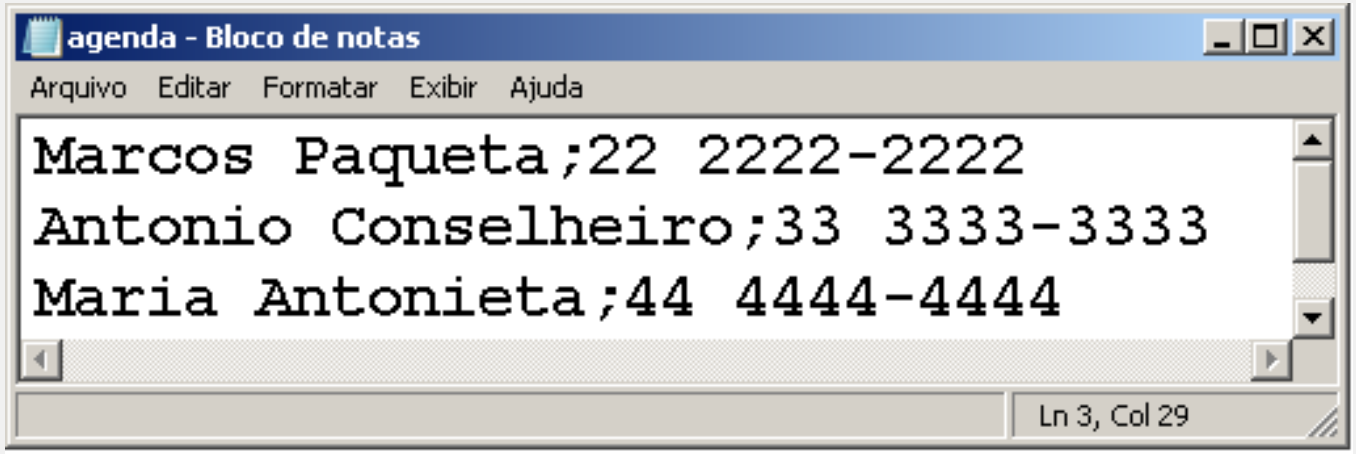
UDESC

Tipos de arquivos

- Os arquivos podem ser classificados em:
  - Arquivos Texto
  - Arquivos Binário

Tipos de arquivos

- Arquivos Texto:
  - são **compostos** por uma **série** de caracteres ASCII **agrupados** em **uma** ou **mais linhas**. Fácil de serem compreendidos pelos seres humanos.



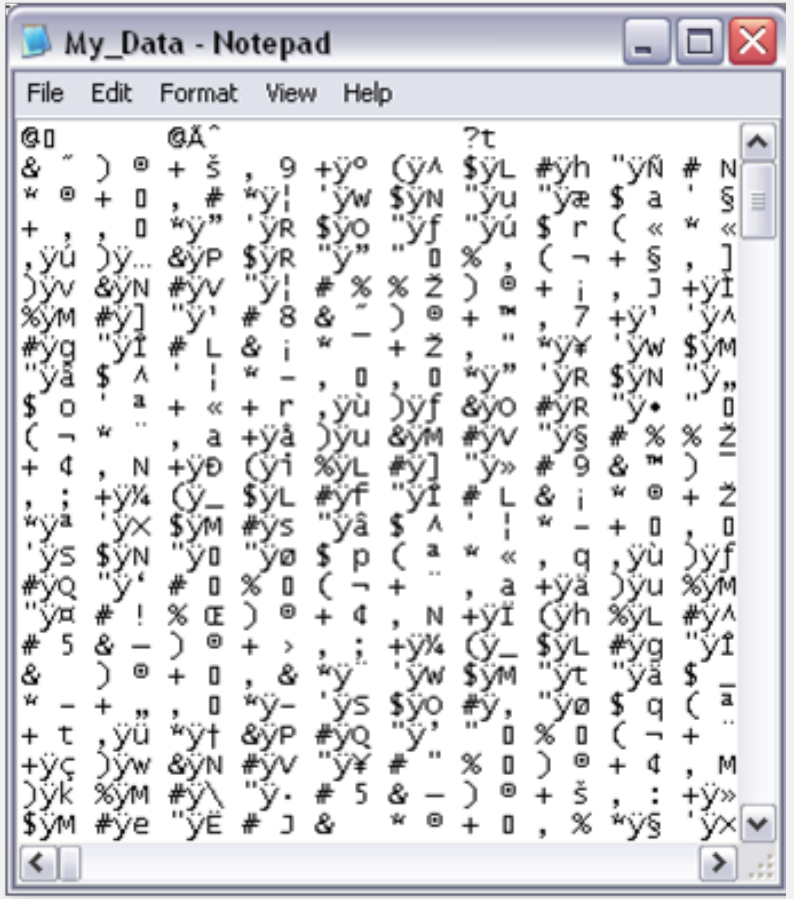
UDESC

Tipos de arquivos

- Arquivo Binário:
  - **compostos** por uma série de bytes **representados** por **caracteres** não compreendidos por **humanos**. São **menores** que os **arquivos** de **texto**.

  - Exemplos:
    - programa de computador
    - arquivo de imagem digital
    - arquivo de som
    - biblioteca compartilhada
    - arquivo de dados

  - Qualquer arquivo que não esteja em formato de texto, é considerado um arquivo binário.



UDESC

API: JAVA/IO

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

java.io

- **Possui** as **classes** para a manipulação de arquivos (import java.io.*;);
- Estas **classes** são **divididas** em duas hierarquias **de acordo** com o **tipo** de **arquivos** que manipulam:
  - *FileInputStream/FileOutputStream* (**arquivos binários**);
  - *FileReader/FileWriter* (**arquivos texto**)
- Os **arquivos** e **diretórios** podem ser **representados** por meio da classe File;
- Documentação:
  - Pacote: Java 8
  - Classe File: Java 8

Classe File

- **Implementa** uma representação abstrata de **nomes** e **caminhos** de arquivos e diretórios no **sistema.**
- **Possibilita** a manipulação de **arquivos** e **diretórios.**
- **Construtores**:
  - *public File(String **pathname**)*
  - *public File(File **parent**, String **child**)*
  - *File(String **parent**, String **child**)*
  - *File(URI **url**)*

Alguns métodos importantes

- public *String* **getParent()**: retorna o diretório pai;
- public **list()**: retorna lista de arquivos contidos no diretório;
- public *boolean* **isFile()**: retorna *true* se é um arquivo;
- public *boolean* **isDirectory()**: retorna *true* se é um diretório;
- public *boolean* **delete()**: tenta apagar o diretório ou arquivo;
- public *boolean* **mkdir()**: cria um diretório com o nome do arquivo;
- public *String* **getAbsolutePath()**: retorna o caminho absoluto;
- public *String* **getPath()**: retorna o caminho do arquivo;
- public *long* **length()**: retorna o tamanho do arquivo em bytes;

Lembre-se

- Embora seja importante conhecermos **nomes** e **métodos** das **classes mais utilizadas**, o interessante aqui é que você enxergue que todos os conceitos previamente estudados **são aplicados a toda hora** nas classes da biblioteca padrão.

Lembre-se

- Não se preocupe em decorar nomes. Atenha-se em entender como essas classes estão relacionadas e como elas estão tirando proveito do uso de **interfaces**, **polimorfismo**, **classes abstratas** e **encapsulamento**. Lembre-se de estar com a documentação (*javadoc*) aberta durante o contato com esses pacotes.

Exemplo 1

- Classe: **ManipulacaoArquivos**
- Método: **criaDiretorios**
- Pacote: **br.udesc.ceplan.jacksonpradolima.exemplosaula.Aula02**

Classe FileReader

- É uma **classe** utilizada para **leitura** em arquivos de texto
- Os **construtores** desta **classe** já assumem uma codificação padrão de **caracteres** (ex: unicode UTF-8) e tamanho de *buffer* **apropriados**
- **Construtores**:
  - *public FileReader(String name)*
  - *public FileReader(File file)*

Alguns métodos importantes

- public *int* **read()**: permite ler um caractere por vez;
- public *boolean* **ready()**: retorna *true* se um arquivo estiver pronto para ser lido;
- public *long* **skip(*long* n)**: pula caracteres em um arquivo, onde *n* é o numero de caracteres;
- public abstract *void* **close()**: fecha um arquivo para permitir acesso com outros métodos; etc.

Exemplo 2

- Classe: **ManipulacaoArquivos**
- Método: **leArquivo**
- Pacote: **br.udesc.ceplan.jacksonpradolima.exemplosaula.Aula02**

Classe BufferedReader

- Para **agilizar** a **leitura** é usada a **classe** chamada *BufferedReader*:

- **Cria** um *buffer* e disponibiliza **métodos** para a leitura de *arrays*, **sequencia** de caracteres e linhas **inteiras.**

Alguns métodos importantes

- public *String* **readLine()**: para ler uma linha de texto, considerando a linha terminada com '\n' (avanço de linha) ou '\r' (retorno de carro – tecla ENTER);
- public *long* **skip(*long* n)**: pula caracteres, tendo como *n* o número de caracteres a serem pulados;
- public *boolean* **ready()**: verifica se o fluxo está pronto para ser lido, ou seja, se o *buffer* não está vazio;
- public *void* **close()**: encerra o fluxo; etc.

Exemplo 3

- Classe: **ManipulacaoArquivos**
- Método: **leArquivoBuffer**
- Pacote: **br.udesc.ceplan.jacksonpradolima.exemplosaula.Aula02**

Classe FileWriter

- É uma **classe** utilizada para **escrita** em arquivos de texto
- **Construtores**:
  - *public FileWriter(String name)*
  - *public FileWriter(String name, boolean append)*
  - *public FileWriter(File file)*
  - *public FileWriter(File file, boolean append)*

Alguns métodos importantes

- public *void* **write(int c)**: permite escrever um caractere por vez;
- public abstract *void* **flush()**: força a liberação do arquivo, de tudo o que foi salvo em *buffer* – de vários métodos write(), imediatamente gravando no destino. Ex.: na hora de fechar um arquivo, para garantir a gravação destes dados que estão no *buffer*;
- public abstract *void* **close()**: fecha o fluxo, liberando-o primeiro (mas não garante gravação do que está no *buffer*); etc.

Exemplo 4

- Classe: **ManipulacaoArquivos**
- Método: **escreveInfo**
- Pacote: **br.udesc.ceplan.jacksonpradolima.exemplosaula.Aula02**

Classe BufferedWriter

- Para **agilizar** a **escrita** é utilizada a **classe** *BufferedWriter*

- **Cria** um *buffer* e disponibiliza **métodos** para a escrita de *arrays*, **sequencia** de caracteres e linhas **inteiras**

Exemplo 5

- Classe: **ManipulacaoArquivos**
- Método: **escreveInfoBuffer**
- Pacote: **br.udesc.ceplan.jacksonpradolima.exemplosaula.Aula02**

Classe FileInputStream

- **Utilizada** para **leitura** de arquivos binários, onde os *bytes* de entrada são **obtidos** a partir de um arquivo em um **sistema de arquivos**
- **Construtores**:
  - *public FileInputStream(String name)*
  - *public FileInputStream(File file)*
- Esta **classe** é **utilizada** principalmente para a leitura de aquivos como **imagens**, **áudio** e **vídeo**

Alguns métodos importantes

- public *int* **available()**: retorna o número de *bytes* que podem ser lidos de um fluxo de entrada de arquivo;
- public *void* **close()**: fecha o fluxo de entrada do arquivo e libera os recursos do sistema associados a ele;
- public *int* **read(*byte*[] b)**: pode ler *b.length bytes* de dados a partir do fluxo de entrada, em um *array* de *bytes*;
- public *long* **skip(*long* n)**: pula e descarta *n bytes* de dados do fluxo de entrada; etc.

Exemplo 6

- Classe: **ManipulacaoArquivos**
- Método: **leInputStream**
- Pacote: **br.udesc.ceplan.jacksonpradolima.exemplosaula.Aula02**

Classe FileOutputStream

- É uma **classe** utilizada para **escrita** em arquivos binários
- **Construtores**:
  - *public FileOutputStream(String name)*
  - *public FileOutputStream(String name, boolean append)*
  - *public FileOutputStream(File file)*
  - *public FileOutputStream(File file, boolean append)*

Alguns métodos importantes

- public *void* **close()**: fecha o fluxo de saída do arquivo e libera os recursos do sistema associados a ele;
- public *void* **write(*int* b)**: grava o *byte* especificado para o fluxo de saída do arquivo;
- public *void* **write(*byte*[] b)**: grava *bytes b.length* a partir de um *array* de *bytes* especificado para o fluxo de saída de arquivo;
- protected *void* **finalize()**: limpa a conexão com o arquivo, garantindo que o método close() seja chamado quando não há mais referências ao fluxo.

Exemplo 7

- Classe: **ManipulacaoArquivos**
- Método: **escreveOutputStream**
- Pacote: **br.udesc.ceplan.jacksonpradolima.exemplosaula.Aula02**

Exemplo 8

- Classe: **ManipulacaoArquivos**
- Método: **escreveOutputStreamB**
- Pacote: **br.udesc.ceplan.jacksonpradolima.exemplosaula.Aula02**

Interface Serializable

- **Java permite** a gravação direta de **objetos** em **disco ou** seu **envio** através da **rede** (ou seja, será serializado – transformado em bytes)
  - Para isso, o **objeto** deve **implementar** a **interface** *java.io.Serializable*
- Um **objeto** que **implementar** a **interface** *Serializable* poderá ser **gravado** em **qualquer** *stream (obj. transm. dados)* **usando** o **método** *writeObject()* da classe *ObjectOutputStream*
  - Também **poderá** ser **recuperado** de **qualquer** *stream* **usando** o **método** *readObject()* da classe *ObjectInputStream*

Interface Serializable

- A **serialização** resume-se em: salvar, gravar, capturar o último estado de um **objeto**.
  - Ou seja, **gravar** em um arquivo o **estado** de um **objeto** de uma classe qualquer para uso futuro
- Para a **serialização** de um objeto ou de uma variável de instância (ou seja, var. que assume o valor de um objeto), é **necessário** que a classe implemente a interface *Serializable*
  - Obs.: *Subclasses* de uma **classe pai** *serializable* implicitamente também já **implementam** a **interface**

Exemplo 9

- Classes: **Aluno** e **AlunoMain**
- Pacote: **br.udesc.ceplan.jacksonpradolima.exemplosaula.Aula02**

EXERCÍCIOS

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Exercícios

1 - **Testar** o **funcionamento** de todos os exemplos **apresentados** na aula de hoje.

2 – [Entregar] Faça uma agenda simples apenas para **armazenar** um **tamanho variável** de contatos, contendo:

- **Nome, Telefone e E-mail;**
- Cada contato deve ser salvo no mesmo arquivo de dados como um objeto;
- Desenvolver procedimentos para recuperar os dados do arquivo.
- O programa deve permitir: Consultar, Inserir, Deletar e Alterar dados.

Leitura Complementar

- Caelum Java/IO
- Manipulando arquivos com recursos do Java 8
- Manipulando Arquivo Txt com Java
- FileWriter e FileOutputStream: quando devo trabalhar com cada um deles?

Obrigado

jacksonpradolima.github.io
github.com/ceplan

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA