# Trabalhando com arquivos

## Programação Orientada a Objetos — QXD0007


UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

Prof. Atílio Gomes Luiz

gomes.atilio@ufc.br

Universidade Federal do Ceará

2º semestre/2021

# Leitura para esta aula



- **Capítulo 15** (Arquivos, fluxos e serialização de objetos) do livro Java Como Programar, Décima Edição.

# Objetivos



- Fazer leitura e escrita sequencial em:
  - arquivo texto
  - arquivo binário
- Serializar e desserializar objetos em arquivos binários
- Fazer leitura e escrita randômica em arquivos binários

# Introdução

# Fluxos de Entrada e Saída



- Input (Entrada) é a transferência de dados de algum dispositivo externo para a memória principal (RAM)

- Output (Saída) é a transferência de dados da memória principal para um dispositivo externo.

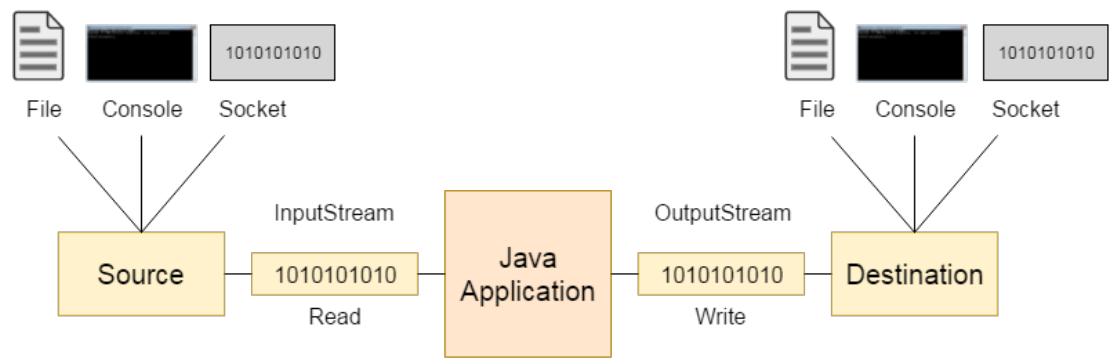
# Fluxos de Entrada e Saída



- Input (Entrada) é a transferência de dados de algum dispositivo externo para a memória principal (RAM)

- Output (Saída) é a transferência de dados da memória principal para um dispositivo externo.

- A fim de realizar operações de Input/Output, um canal de comunicação, chamado stream, é estabelecido entre o dispositivo e a RAM.
  - Stream (fluxo) é uma sequência de bytes usada como origem ou destino dos dados consumidos ou gerados por um programa.



# Fluxos de Entrada e Saída



Não importa de onde os dados vêm ou para onde eles vão e não importa o tipo, pois os algoritmos para ler e gravar dados sequencialmente são basicamente os mesmos:

**Leitura**

abra uma stream
**enquanto** tiver informação
    leia a informação
feche a stream

**Escrita**

abra uma stream
**enquanto** tiver informação
    escreva a informação
feche a stream

# Tipos de fluxos



- As streams podem ser utilizadas para entrada e saída de dados como bytes ou caracteres.

- **fluxos baseados em bytes:** leem e escrevem o dado em seu formato binário: um `char` é dois bytes, um `int` é quatro bytes, etc.
  - Um arquivo criado usando esse tipo de fluxo é chamado `arquivo binário` e só pode ser lido por um programa que entende o conteúdo específico do arquivo.

# Tipos de fluxos



- As streams podem ser utilizadas para entrada e saída de dados como bytes ou caracteres.

- **fluxos baseados em bytes:** leem e escrevem o dado em seu formato binário: um `char` é dois bytes, um `int` é quatro bytes, etc.
  - Um arquivo criado usando esse tipo de fluxo é chamado `arquivo binário` e só pode ser lido por um programa que entende o conteúdo específico do arquivo.

- **fluxos baseados em caracteres:** leem e escrevem o dado como uma sequência de caracteres na qual cada caracter tem dois bytes — o número de bytes de um dado valor depende do número de caracteres naquele valor.
  - Um arquivo criado usando esse tipo de fluxo é chamado `arquivo texto` e pode ser lido por editores de texto.

# Fluxos de Entrada/Saída Padrão



Ao executar um programa, o sistema operacional estabelece 3 streams de bytes:

1. **standard output stream**: é o fluxo de saída padrão, que mostra a saída em forma de caracteres via console. No Java, esse stream pode ser acessado através do objeto `System.out`. De fato, `System.out` é uma referência a um objeto da classe `OutputStream`.

# Fluxos de Entrada/Saída Padrão



Ao executar um programa, o sistema operacional estabelece 3 streams de bytes:

1. **standard output stream**: é o fluxo de saída padrão, que mostra a saída em forma de caracteres via console. No Java, esse stream pode ser acessado através do objeto `System.out`. De fato, `System.out` é uma referência a um objeto da classe `OutputStream`.

2. **standard input stream**: é o fluxo de entrada padrão, que permite ler dados do teclado. No Java, esse stream pode ser acessado através do objeto `System.in`. Esse objeto pertence à classe `InputStream`.

# Fluxos de Entrada/Saída Padrão



Ao executar um programa, o sistema operacional estabelece 3 streams de bytes:

1. **standard output stream**: é o fluxo de saída padrão, que mostra a saída em forma de caracteres via console. No Java, esse stream pode ser acessado através do objeto `System.out`. De fato, `System.out` é uma referência a um objeto da classe `OutputStream`.

2. **standard input stream**: é o fluxo de entrada padrão, que permite ler dados do teclado. No Java, esse stream pode ser acessado através do objeto `System.in`. Esse objeto pertence à classe `InputStream`.

3. **standad error stream**: é o fluxo de erro padrão. Envia mensagens de erro como fluxo de caracteres para a saída padrão. No Java, esse stream pode ser acessado através do objeto `System.err`. Esse objeto pertence à classe `OutputStream`.

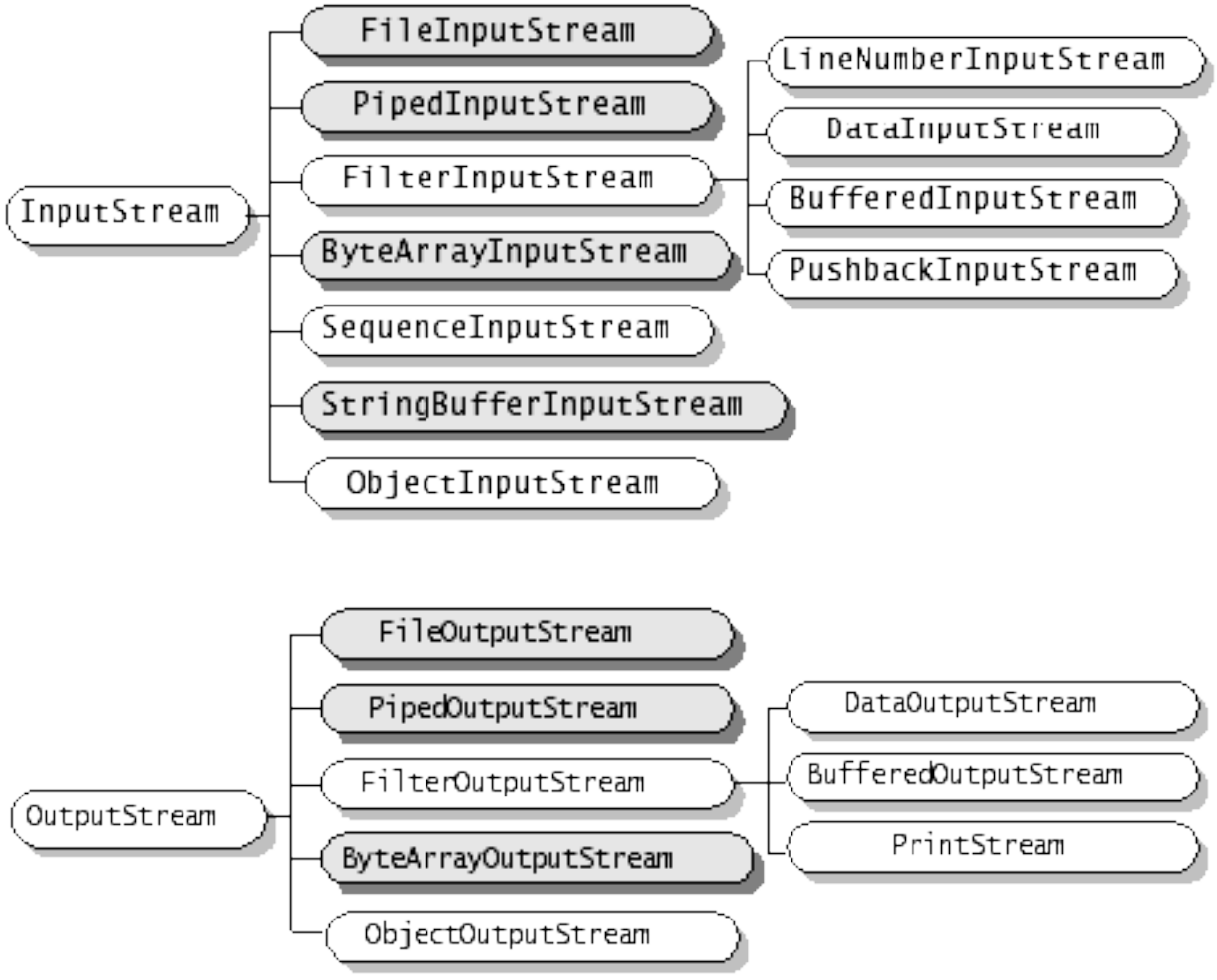
# Byte Streams



- O pacote `java.io` possui duas classes abstratas, `InputStream` e `OutputStream`, que definem o comportamento padrão das streams de entrada e saída de bytes, respectivamente, em Java.

- Esse pacote se vale do polimorfismo a fim de executar I/O
  - O pacote java.io utiliza streams de entrada (`InputStream`) e de saída (`OutputSteam`) para toda e qualquer operação, seja ela relativa a um **arquivo**, a um campo **blob** do banco de dados, a uma conexão remota via **sockets**, ou até mesmo às **entradas e saída padrão** de um programa.

# InputStream e OutputStream




Classes em cinza leem ou escrevem dados em coletores de dados, as demais classes realizam alguma forma de processamento.

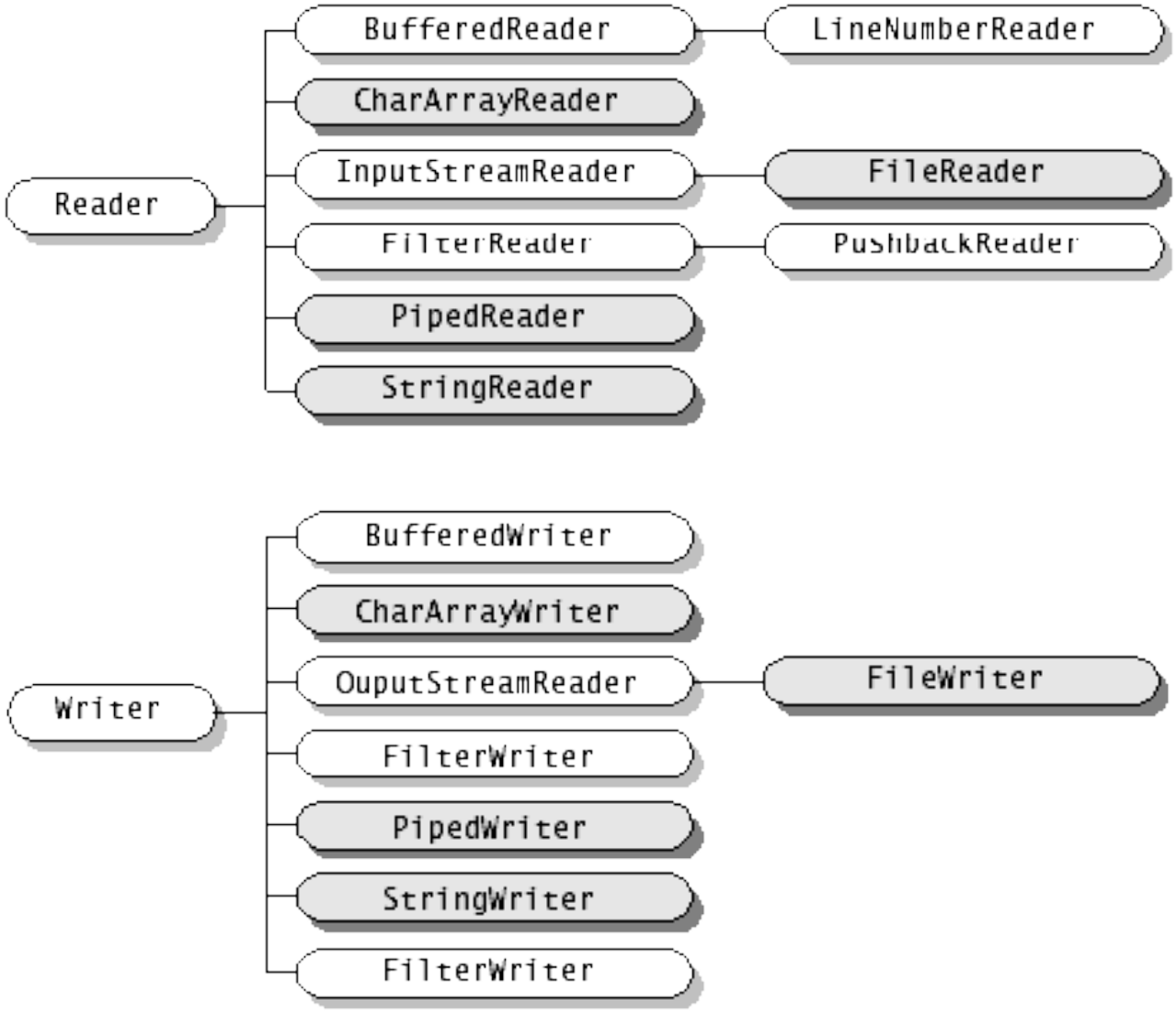
# Character Streams



- Problema: Byte streams são inconvenientes para processar informação armazenada em Unicode (lembre que Unicode usa múltiplos bytes por caractere)

- Portanto, o pacote `java.io` possui duas classes abstratas, `Reader` e `Writer`, que definem o comportamento padrão das streams de entrada e saída de caracteres em Java.

# Reader e Writer



# Lendo e escrevendo em um arquivo texto (à moda antiga)

# Leitura de arquivo



Podemos ler um arquivo texto em três etapas:

- Primeiro, instanciamos um objeto da classe `FileInputStream`, que é uma subclasse de InputStream. Instanciamos esse objeto passando o nome do arquivo ou o caminho absoluto de onde o arquivo está. O `FileInputStream` lê o arquivo byte-a-byte.

- Para recuperar um caractere, precisamos traduzir os bytes com o encoding dado para o respectivo código unicode, e isso pode usar um ou mais bytes. Quem vai fazer isso é um objeto da classe `InputStreamReader`. Criamos esse objeto passando para o seu construtor o `FileInputStream` criado no passo anterior.

- Apesar do objeto `InputStreamReader` já ajudar no trabalho de manipulação de caracteres, ainda será difícil pegar uma string. A classe `BufferedReader` é um `Reader` que recebe outro `Reader` pelo construtor e concatena os diversos chars para formar uma `String`.

# Leitura de arquivo — Exemplo



- Analisar os arquivos do Projeto01.

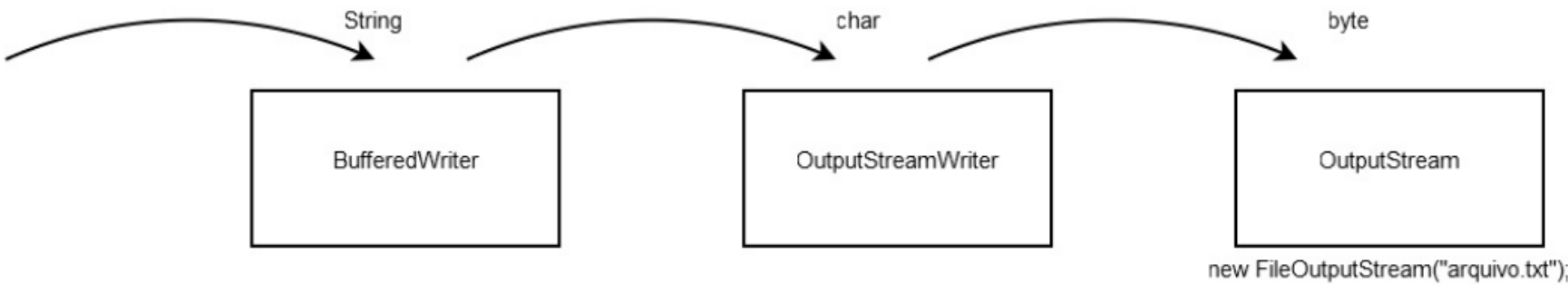
# Escrita em arquivo

Universidade Federal do Ceará
Campus Quixadá

Podemos escrever um arquivo texto em três etapas:

- A fim de escrever em um arquivo texto, podemos utilizar as classes `FileOutputStream`, `OutputStreamWriter` e `BufferedWriter`.

- Escrever um arquivo é o processo inverso da leitura:




- Analisar os arquivos do Projeto2.

# Arquivos Texto

# Escrita em arquivo texto sequencial



- A fim de escrever dados para um arquivo texto, vamos criar objetos das seguintes classes:
  - `FileWriter` — permite que você abra um arquivo texto para gravar caracteres. Esta classe permite escolher se o conteúdo já existente no arquivo aberto será apagado ou preservado.
  - `PrintWriter` — permite gravar dados de qualquer tipo primitivo ou objetos String em um arquivo texto. Esta classe possui métodos de formatação tais como print e println, que nos permitem escrever longas strings na stream de saída. Por esse motivo, essa classe é usada em conjunto com um objeto `FileWriter` que tem uma conexão com um arquivo.

# Arquivos Texto — Criação



- Um objeto `PrintWriter` "empacota" um objeto `FileWriter` a fim de gravar dados em um arquivo texto.

```
FileWriter fstream = new FileWriter(‘‘info.txt’’);
PrintWriter outputFile = new PrintWriter(fstream);
```

- Se o arquivo `info.txt` já existir, então o objeto `FileWriter` irá apagá-lo e um novo arquivo com o mesmo nome será criado.

# Arquivos Texto — Criação



- Um objeto `PrintWriter` "empacota" um objeto `FileWriter` a fim de gravar dados em um arquivo texto.

```
FileWriter fstream = new FileWriter(‘‘info.txt’’);
PrintWriter outputFile = new PrintWriter(fstream);
```

- Se o arquivo `info.txt` já existir, então o objeto `FileWriter` irá apagá-lo e um novo arquivo com o mesmo nome será criado.

- Caso você queira que os dados anteriores sejam mantidos e os novos sejam inseridos ao final do arquivo, basta usar o outro construtor:

```
FileWriter fstream = new FileWriter(‘‘info.txt’’, true);
PrintWriter outputFile = new PrintWriter(fstream);
```

# Arquivos Texto — Criação



- A classe `FileWriter` possui 9 construtores, alguns exibidos abaixo:
  - `FileWriter(String fileName)`
  - `FileWriter(String fileName, boolean append)`
  - `FileWriter(String filename, Charset charset)`
  - `FileWriter(String filename, Charset charset, boolean append)`

- Os construtores de `FileWriter` lançarão uma `IOException` se o arquivo não puder ser encontrado.

- Uma vez que o objeto `PrintWriter` for criado, você pode usá-lo para escrever dados no arquivo binário.

# Arquivos Texto — Escrita

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

A classe `PrintWriter` possui diversos métodos para escrita de dados em arquivos texto.

- `PrintWriter` possui versões sobrecarregadas do método `println` para cada um dos tipos primitivos, para Strings e para Objects. Esse método converte o dado para string e o grava na stream de saída seguido do caractere de quebra de linha.

- `PrintWriter` também possui versões sobrecarregadas do método `print` para cada um dos tipos primitivos, para Strings e para Objects.

- `void printf(String format, Object... args)`: Grava uma string formatada usando a string de formato e os argumentos especificados.

- Consulte a API: `https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/PrintWriter.html`

# Arquivos Texto — Escrita



- A classe `PrintWriter` é uma classe "buferizada".
- Se o objeto PrintWriter estiver setado no modo `autoflushing`, todos os caracteres no buffer serão enviados para o destino sempre que os métodos `println`, `printf` e `print` forem invocados.
  - Por default, o `autoflushing` não é habilitado.
- Você pode habilitar o `autoflushing` usando o segundo construtor abaixo:
  - `PrintWriter(Writer out)`
  - `PrintWriter(Writer out, boolean autoFlush)`

# Exemplo de Escrita em Arquivo Texto



- Analisar o arquivo `EscritaTexto.java`

# Arquivos Texto — Leitura



- A fim de abrir um arquivo texto para leitura, vamos criar objetos das classes `File` e `Scanner`.

```
File file = new File(‘‘info.txt’’);
Scanner inputFile = new Scanner(file);
```

# Arquivos Texto — Leitura



- A fim de abrir um arquivo texto para leitura, vamos criar objetos das classes `File` e `Scanner`.

```
File file = new File(‘‘info.txt’’);
Scanner inputFile = new Scanner(file);
```

- Se o arquivo não puder ser aberto, uma `FileNotFoundException` será lançada pelo construtor da classe `Scanner`.

- Uma vez que o objeto `Scanner` for criado, você pode usá-lo para ler dados no arquivo texto, de modo semelhante ao que fazemos quando lemos do entrada de dados padrão `System.in`.

# Arquivos Binários — Leitura



A classe `Scanner` possui diversos métodos para leitura de dados em arquivos texto.

- `boolean hasNext()`: retorna `true` se ainda houver bytes a serem lidos na stream de entrada.
- `String next()`: retorna a próxima string
- `String nextLine()`: Este método retorna o resto da linha atual, excluindo qualquer separador de linha no final. A posição é definida para o início da próxima linha.
- Consulte a API: `https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Scanner.html`

# Exemplo de Leitura em Arquivo Binário



- Analisar o arquivo `LeituraTexto.java`

# Exemplo Projeto

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- `ProjetoReadWriteTextFile`

# Arquivos Binários

# Arquivos Binários



- O modo como os dados são armazenados na memória é algumas vezes chamado formato binário bruto.
  - Dados em formato binário bruto podem ser armazenados em um arquivo, denominado arquivo binário.

- Armazenar dados em seu formato binário é mais eficiente do que armazená-los em modo texto.

- Arquivos binários não podem ser lidos por um editor de texto e, geralmente, só são inteligíveis pela a aplicação que os criou.

# Arquivos Binários — Criação



- A fim de escrever dados para um arquivo binário, você deve criar objetos das seguintes classes:
  - `FileOutputStream` — permite que você abra um arquivo para gravar dados binários. Esta classe fornece apenas funcionalidades básicas para gravar bytes no arquivo.
  - `DataOutputStream` — permite gravar dados de qualquer tipo primitivo ou objetos String em um arquivo binário. Porém, não consegue acessar um arquivo diretamente. Por esse motivo, essa classe é usada em conjunto com um objeto `FileOutputStream` que tem uma conexão com um arquivo.

# Arquivos Binários — Criação



- Um objeto `DataOutputStream` "empacota" um objeto `FileOutputStream` a fim de gravar dados em um arquivo binário.

```
FileOutputStream fstream =
    new FileOutputStream(‘‘MyInfo.dat’’);
DataOutputStream outputFile =
    new DataOutputStream(fstream);
```

- Se o arquivo `MyInfo.dat` já existir, então o objeto `FileOutputStream` irá apagá-lo e um novo arquivo com o mesmo nome será criado.

# Arquivos Binários — Criação



- Um objeto `DataOutputStream` "empacota" um objeto `FileOutputStream` a fim de gravar dados em um arquivo binário.

```
FileOutputStream fstream =
    new FileOutputStream(‘‘MyInfo.dat’’);
DataOutputStream outputFile =
    new DataOutputStream(fstream);
```

- Se o arquivo `MyInfo.dat` já existir, então o objeto `FileOutputStream` irá apagá-lo e um novo arquivo com o mesmo nome será criado.

- Caso você queira que os dados anteriores sejam mantidos e os novos sejam inseridos ao final do arquivo, basta usar o outro construtor:

```
FileOutputStream fstream =
    new FileOutputStream(‘‘MyInfo.dat’’, true);
DataOutputStream outputFile =
    new DataOutputStream(fstream);
```

# Arquivos Binários — Criação



- A classe `FileOutputStream` possui alguns construtores:
  - `FileOutputStream(String name)`
  - `FileOutputStream(String name, boolean append)`
  - `FileOutputStream(File file)`
  - `FileOutputStream(File file, boolean append)`

- Os construtores de `FileOutputStream` lançarão uma `FileNotFoundException` se o arquivo não puder ser encontrado.

- Uma vez que o objeto `DataOutputStream` for criado, você pode usá-lo para escrever dados no arquivo binário.

# Arquivos Binários — Escrita



A classe `DataOutPutStream` possui diversos métodos para escrita de dados em arquivos binários.

- `void writeUTF(String str)`: escreve uma string na stream de saída usando uma versão modificada do formato UTF-8.
  - Antes de escrever a string, este método escreve um inteiro de dois bytes indicando o número de bytes que a string ocupa.
  - Em seguida, ele grava os caracteres da string em Unicode. (UTF significa Unicode Text Format).

# Arquivos Binários — Escrita



A classe `DataOutPutStream` possui diversos métodos para escrita de dados em arquivos binários.

- `void writeUTF(String str)`: escreve uma string na stream de saída usando uma versão modificada do formato UTF-8.
  - Antes de escrever a string, este método escreve um inteiro de dois bytes indicando o número de bytes que a string ocupa.
  - Em seguida, ele grava os caracteres da string em Unicode. (UTF significa Unicode Text Format).

- `void writeDouble(double v)`: Converte o `double` para `long` usando o método `doubleToLongBits` da classe `Double` e, em seguida, grava esse `long` na stream de saída subjacente.

# Arquivos Binários — Escrita



A classe `DataOutPutStream` possui diversos métodos para escrita de dados em arquivos binários.

- `void writeUTF(String str)`: escreve uma string na stream de saída usando uma versão modificada do formato UTF-8.
  - Antes de escrever a string, este método escreve um inteiro de dois bytes indicando o número de bytes que a string ocupa.
  - Em seguida, ele grava os caracteres da string em Unicode. (UTF significa Unicode Text Format).

- `void writeDouble(double v)`: Converte o `double` para `long` usando o método `doubleToLongBits` da classe `Double` e, em seguida, grava esse `long` na stream de saída subjacente.

- Existem métodos similares para cada tipo nativo. Consulte a API: `https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/DataOutputStream.html`

# Exemplo de Escrita em Arquivo Binário



- Analisar o arquivo `EscritaBinario.java`

# Arquivos Binários — Leitura



- A fim de abrir um arquivo binário para leitura, você deve criar objetos das classes `FileInputStream` e `DataInputStream`.

```
FileInputStream fstream =
    new FileInputStream(‘‘MyInfo.dat’’);
DataInputStream inputFile =
    new DataInputStream(fstream);
```

# Arquivos Binários — Leitura



- A fim de abrir um arquivo binário para leitura, você deve criar objetos das classes `FileInputStream` e `DataInputStream`.

```
FileInputStream fstream =
    new FileInputStream(‘‘MyInfo.dat’’);
DataInputStream inputFile =
    new DataInputStream(fstream);
```

- Se o arquivo não existir, uma `FileNotFoundException` pelo construtor da classe `FileInputStream`.

- Uma vez que o objeto `DataInputStream` for criado, você pode usá-lo para ler dados no arquivo binário.

# Arquivos Binários — Leitura



A classe `DataInputStream` possui diversos métodos para leitura de dados em arquivos binários.

- `String readUTF()`: Lê uma string que foi codificada usando o formato UTF-8. Este método é adequado para ler dados gravados pelo método `writeUTF` da interface `DataOutput`.

# Arquivos Binários — Leitura



A classe `DataInputStream` possui diversos métodos para leitura de dados em arquivos binários.

- `String readUTF()`: Lê uma string que foi codificada usando o formato UTF-8. Este método é adequado para ler dados gravados pelo método `writeUTF` da interface `DataOutput`.

- `double readDouble()`: Lê oito bytes de entrada e retorna o valor em `double`. Este método é adequado para ler bytes gravados pelo método `writeDouble` da interface `DataOutput`.

# Arquivos Binários — Leitura



A classe `DataInputStream` possui diversos métodos para leitura de dados em arquivos binários.

- `String readUTF()`: Lê uma string que foi codificada usando o formato UTF-8. Este método é adequado para ler dados gravados pelo método `writeUTF` da interface `DataOutput`.

- `double readDouble()`: Lê oito bytes de entrada e retorna o valor em `double`. Este método é adequado para ler bytes gravados pelo método `writeDouble` da interface `DataOutput`.

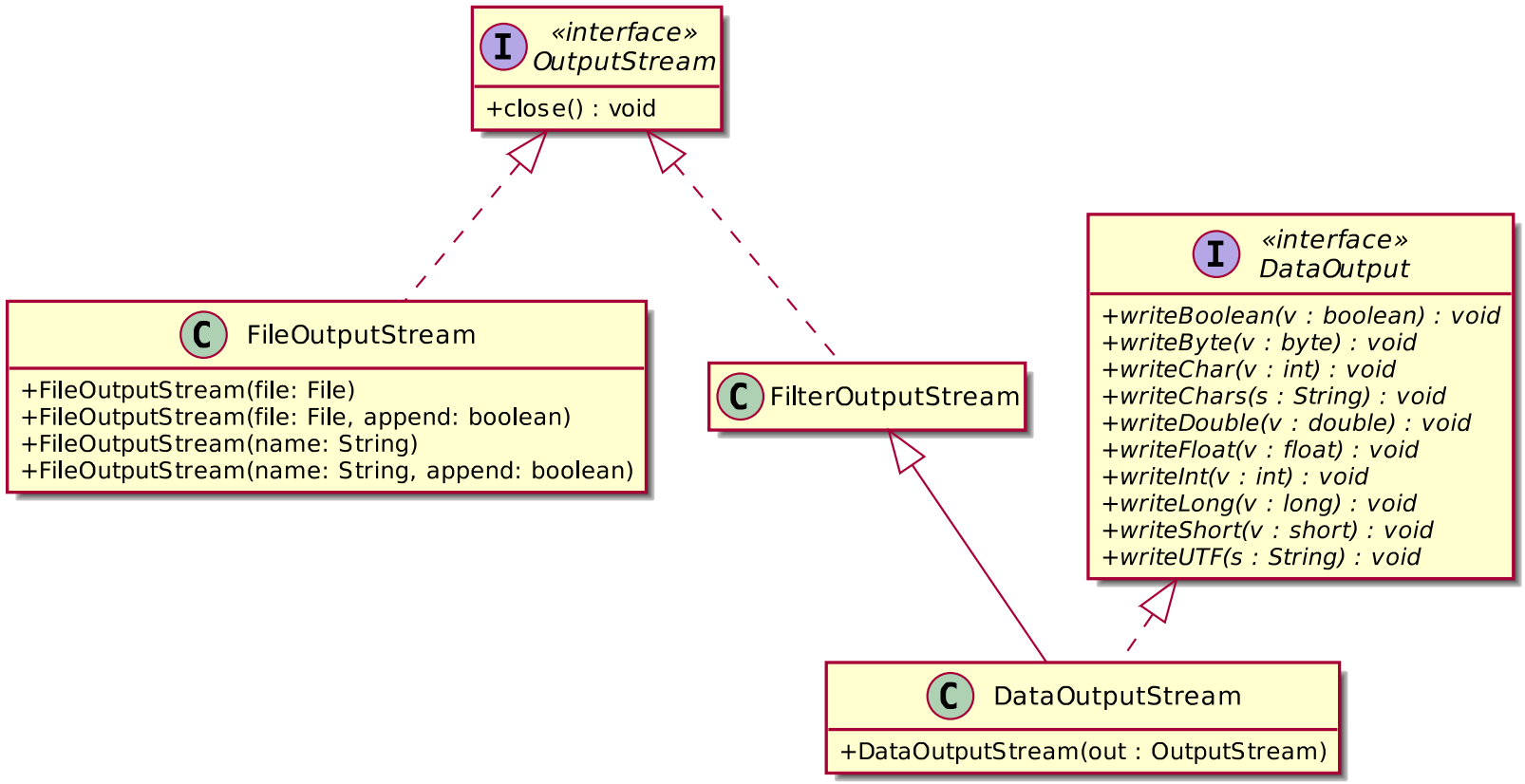
- Existem métodos similares para cada tipo nativo. Consulte a API: `https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/DataInputStream.html`

# Exemplo de Leitura em Arquivo Binário



- Analisar o arquivo `LeituraBinario.java`

# Classes de escrita de arquivo binário

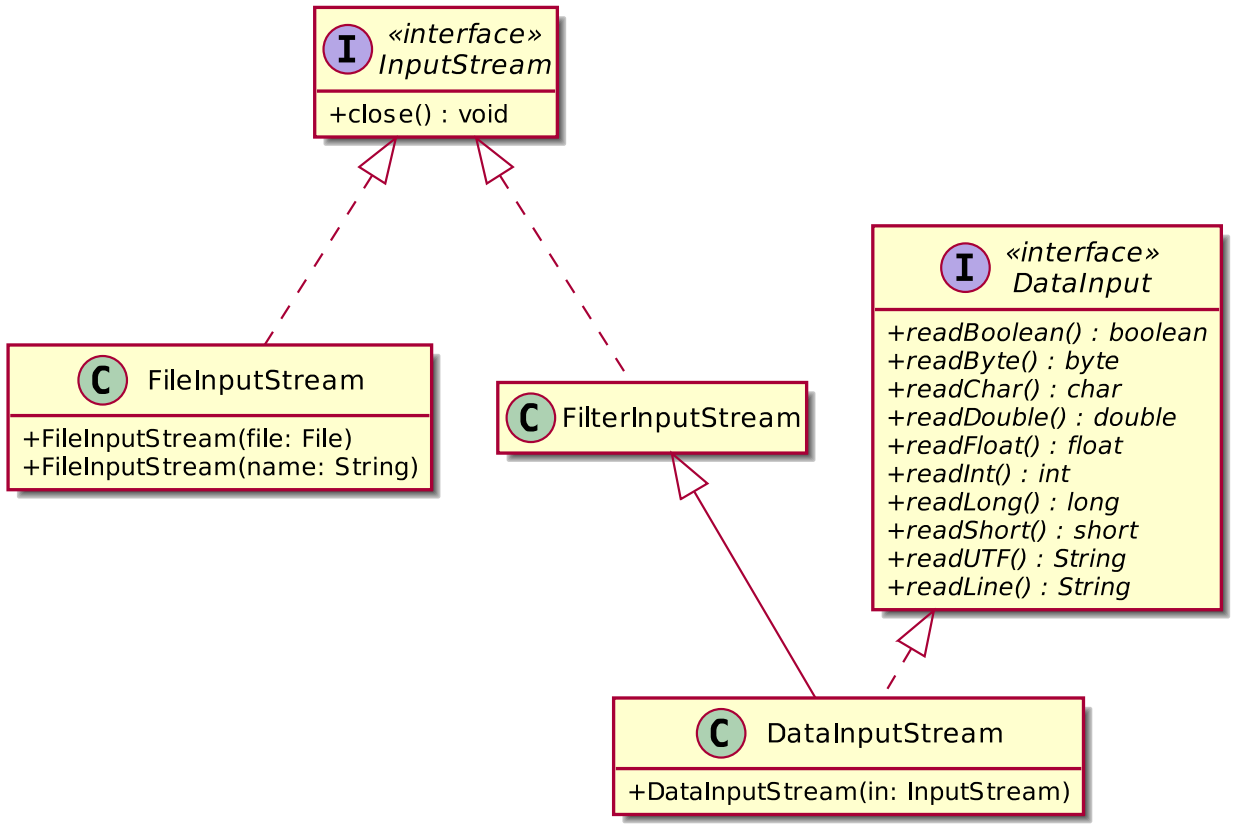



https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/DataOutputStream.html

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/FileOutputStream.html

# Classes de leitura de arquivo binário






https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/DataInputStream.html

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/io/FileInputStream.html

# Exemplo



- `ProjetoReadWriteBinarySequentialFile`

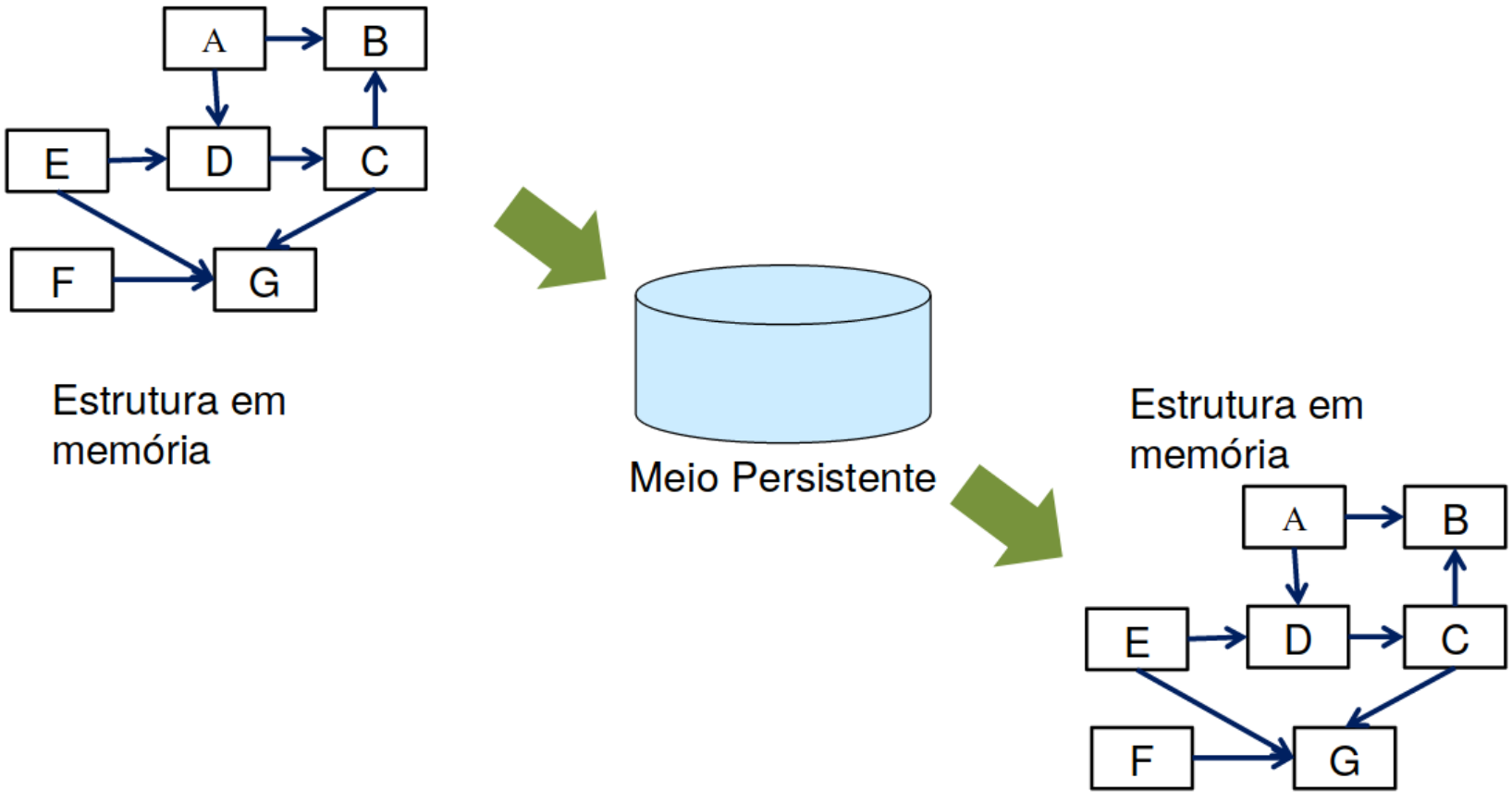
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

# Serialização de Objetos

# Serialização de Objetos



## Problema Geral



# Serialização de Objetos



- **Serialização:** consiste em representar um objeto como uma sequência de bytes.
  - Um **objeto serializado** é um objeto representado como uma sequência de bytes que inclui os dados do objeto assim como as informações sobre o tipo do objeto e os tipos dos dados armazenados no objeto.

# Serialização de Objetos



- **Serialização:** consiste em representar um objeto como uma sequência de bytes.
  - Um **objeto serializado** é um objeto representado como uma sequência de bytes que inclui os dados do objeto assim como as informações sobre o tipo do objeto e os tipos dos dados armazenados no objeto.
  - Um objeto é serializado pode ser transmitido por uma stream e armazenado em disco.

- **Desserialização:** consiste em reconstruir um objeto a partir de uma sequência de bytes.
  - O objeto é lido a partir do arquivo por meio de uma stream.
  - Então, ele é desserializado e reconstruído na memória do computador.

# Serialização de Objetos



- O pacote `java.io` oferece diversas classes de apoio à serialização de objetos.
  - **ObjectOutputStream**: permite a serialização de uma estrutura de objetos num dispositivo de saída. Implementa a interface `ObjectOutput`
  - **ObjectInputStream**: permite a desserialização de objetos a partir de dados lidos de um dispositivo de entrada. Implementa a interface `ObjectInput`

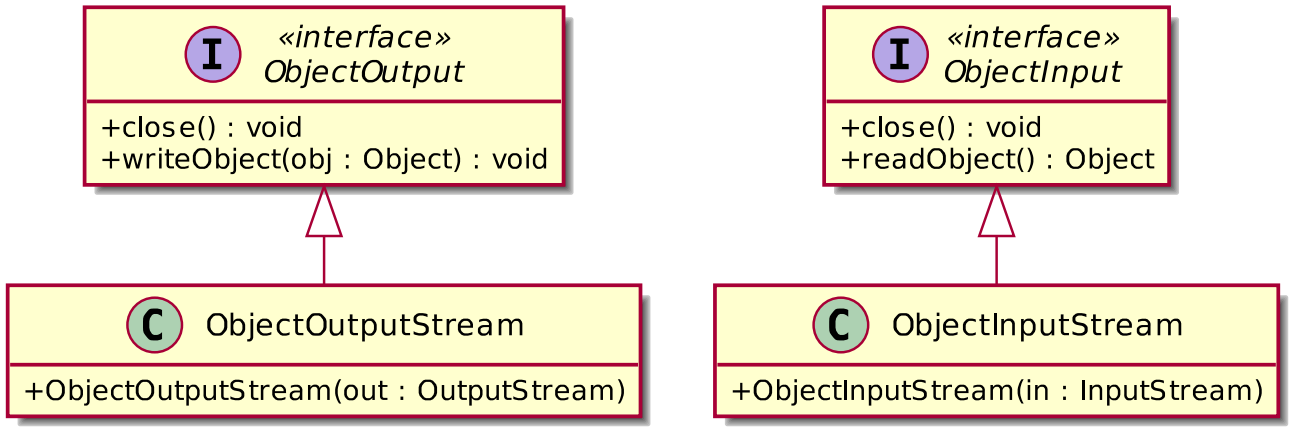
# Serialização de Objetos



- O pacote `java.io` oferece diversas classes de apoio à serialização de objetos.
  - **ObjectOutputStream**: permite a serialização de uma estrutura de objetos num dispositivo de saída. Implementa a interface `ObjectOutput`
  - **ObjectInputStream**: permite a desserialização de objetos a partir de dados lidos de um dispositivo de entrada. Implementa a interface `ObjectInput`
- **ObjectOutputStream** e **ObjectInputStream** simplesmente leem ou escrevem sequências de bytes que representam objetos — eles não sabem de onde ler os bytes ou para onde escrevê-los.
  - Por isso um objeto stream deve ser passado para estas classes. Uma vez de posse desses streams, eles podem ler e escrever os bytes.

# Interfaces `ObjectOutput` e `ObjectInput`

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

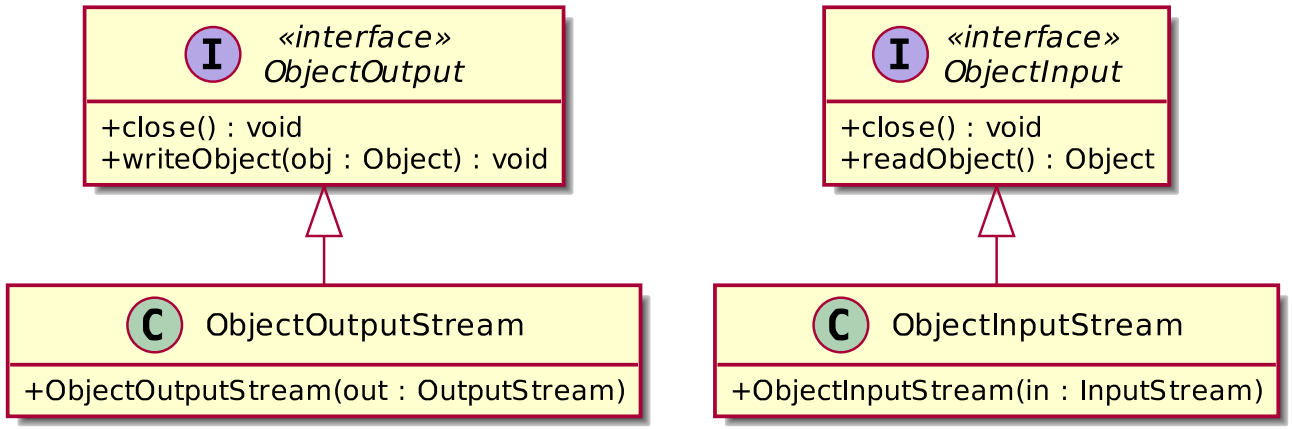



- `ObjectOutputStream` implementa a interface `ObjectOutput`. Essa interface contém o método **writeObject**, que recebe um `Object` como argumento e escreve sua informação em um `OutputStream`.

- `ObjectInputStream` implementa a interface `ObjectInput`, que contém o método **readObject**, que lê um `InputStream` e retorna uma referência para um `Object`.

# Interfaces `ObjectOutput` e `ObjectInput`

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ




- `ObjectOutputStream` implementa a interface `ObjectOutput`. Essa interface contém o método **writeObject**, que recebe um `Object` como argumento e escreve sua informação em um `OutputStream`.

- `ObjectInputStream` implementa a interface `ObjectInput`, que contém o método **readObject**, que lê um `InputStream` e retorna uma referência para um `Object`.

# Interface `Serializable`



- A interface `java.io.Serializable` deve ser implementada pelas classes a serem serializadas.

- Esta interface é apenas de marcação, pois não tem nenhum atributo e nenhum método a ser implementado, serve apenas para que a JVM saiba que aquela determinada classe está hábil para ser serializada.

# Interface `Serializable`



- A interface `java.io.Serializable` deve ser implementada pelas classes a serem serializadas.

- Esta interface é apenas de marcação, pois não tem nenhum atributo e nenhum método a ser implementado, serve apenas para que a JVM saiba que aquela determinada classe está hábil para ser serializada.

- Por padrão, todas as variáveis de tipo nativo e do tipo String são serializáveis. Arrays também são serializáveis. Todas as coleções do Java são serializáveis.
  - Para demais variáveis do tipo referência, você deve checar a documentação da classe e de suas superclasses para determinar se ela é serializável.

# Interface `Serializable`



- Em uma classe `Serializable`, todas os atributos de instância devem ser `Serializable`.

- Toda variável de instância que for não-serializável deve ser declarada `transient` para indicar que elas deveriam ser ignoradas durante o processo de serialização.

# Exemplo



- Analisar `ProjectSerial`
- Analisar `ProjectSerial02`

# Arquivos de acesso aleatório (Random acess files)

# Arquivos de acesso sequencial



- Os arquivos de texto e arquivos binários vistos anteriormente usam acesso sequencial.

- Com o acesso sequencial:
  - quando o arquivo é aberto, os dados são lidos a partir do início do arquivo
  - a medida que a leitura continua, a posição de leitura avança sequencialmente pelo conteúdo do arquivo.

- Se o arquivo for muito grande, tentar localizar um registro nas últimas posições do arquivo usando acesso sequencial pode tomar muito tempo.

# Arquivos de acesso aleatório



- Java permite que um programa execute acesso aleatório a arquivos.
- No acesso aleatório, o programa pode pular imediatamente para qualquer local do arquivo.
- Para criar e trabalhar com arquivos de acesso aleatório em Java, usamos a classe `RandomAccessFile`.
  - Arquivos criados com esta classe são tratados como binários.

# Arquivos de acesso aleatório



- Java permite que um programa execute acesso aleatório a arquivos.
- No acesso aleatório, o programa pode pular imediatamente para qualquer local do arquivo.
- Para criar e trabalhar com arquivos de acesso aleatório em Java, usamos a classe `RandomAccessFile`.
  - Arquivos criados com esta classe são tratados como binários.
- Construtor: `RandomAccessFile(String filename, String mode)`
  - `filename`: o nome do arquivo
  - `mode`: modo no qual você deseja usar o arquivo:
    - "r" — reading
    - "rw" — reading and writing

# Classe RandomAccessFile



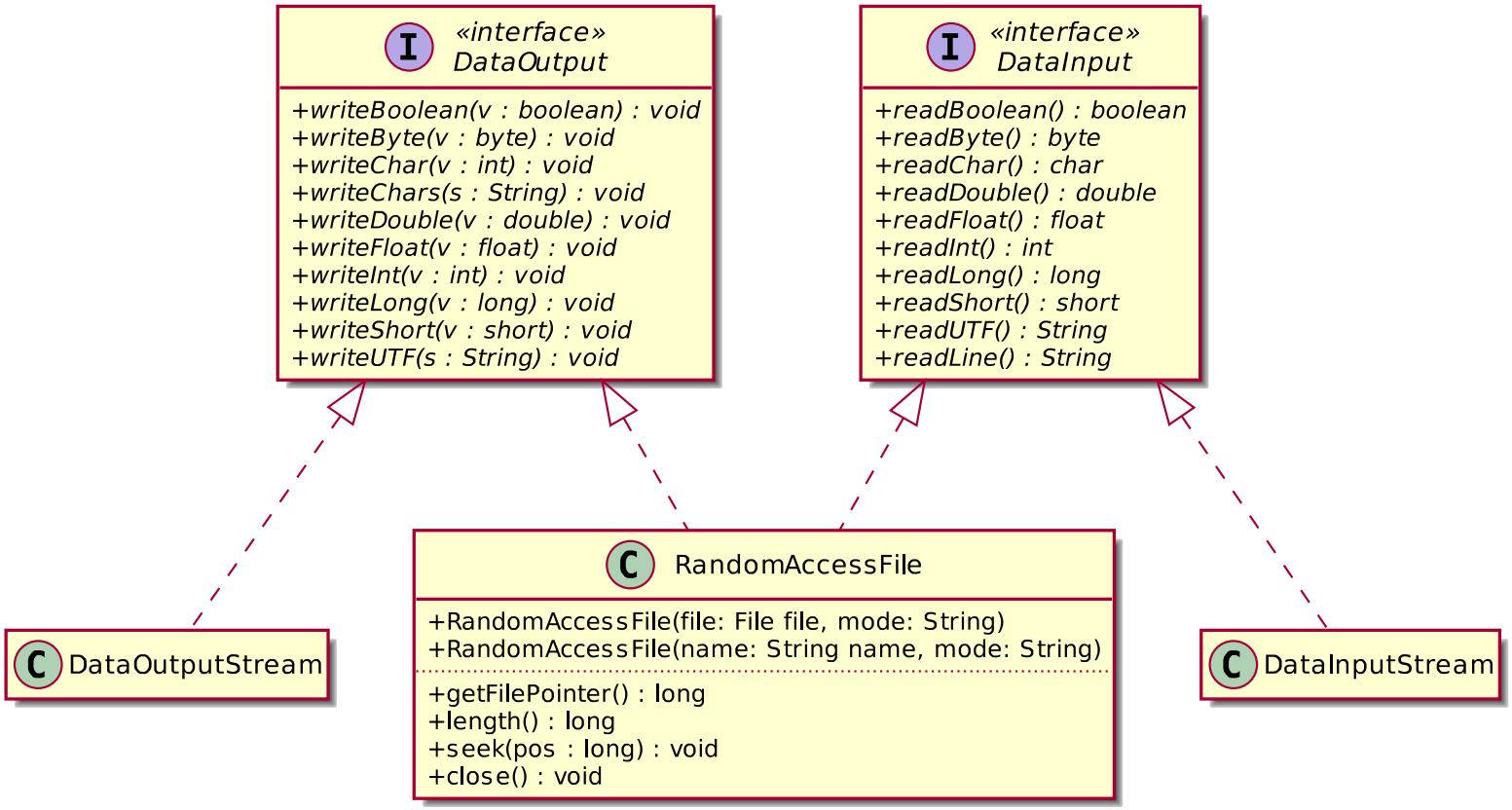
```
// Open a file for random reading
RandomAccessFile rf = new RandomAccessFile("arq.dat", "r");

// Open a file for random reading and writing
RandomAccessFile rf = new RandomAccessFile("arq.dat", "rw");
```

- Ao tentar abrir um arquivo no modo "r" e o arquivo não existir, uma `FileNotFoundException` será lançada.
- Abrir um arquivo no modo "r" e tentar escrever nele irá lançar uma `IOException`.
- Ao tentar abrir um arquivo existente no modo "rw", ele não é deletado e o conteúdo anterior do arquivo é preservado.
- Se você abrir um arquivo no modo "rw" e ele ainda não existe, então ele será criado.

# Classe `RandomAccessFile`






- A classe `RandomAccessFile` tem os mesmos métodos para escrita de dados que a classe `DataOutputStream` e os mesmos métodos para leitura de dados que a classe `DataInputStream`, vistas anteriormente.

# O ponteiro de arquivo (*file pointer*)



- A classe `RandomAcessFile` trata um arquivo como um array de bytes.

- Os bytes são numerados:
  - o primeiro byte possui número 0
  - O número do último byte é o número de bytes do arquivo menos 1.
  - A classe `RandomAcessFile` fornece o método `length()` que retorna o tamanho do arquivo em bytes como um `long`.

- Essa numeração dos bytes é semelhante à indexação de arrays e é usada para identificar posições no arquivo.

- A fim de rastrear a posição atual de leitura e escrita no arquivo, o objeto do tipo `RandomAccessFile` mantém um atributo do tipo `long` conhecido como file pointer.
  - A classe `RandomAcessFile` fornece o método `getFilePointer()` que retorna o valor do ponteiro de arquivo.

# O ponteiro de arquivo (*file pointer*)



- O ponteiro de arquivo guarda a posição atual de onde pode-se ler ou escrever num arquivo.
- Quando um arquivo é aberto, o ponteiro de arquivo é definido como 0.
- Quando um item é lido do arquivo, ele é lido a partir do byte para o qual o ponteiro de arquivo aponta.
- Ler o arquivo faz com que o ponteiro de arquivo avance para o byte logo após o item que foi lido.
- Uma `EOFException` é lançada quando uma leitura faz com que o ponteiro do arquivo ultrapasse o tamanho do arquivo.

# O ponteiro de arquivo (*file pointer*)



- A escrita também ocorre a partir do byte apontado pelo ponteiro de arquivo.
- Se o ponteiro de arquivo apontar para o final do arquivo, os dados serão gravados no final do arquivo.
- Se o ponteiro de arquivo contém o número de um byte dentro do arquivo, em um local onde já existem dados armazenados, uma gravação irá sobrescrever os dados naquele ponto.

# O ponteiro de arquivo (*file pointer*)



- A classe `RandomAccessFile` permite mover o ponteiro de arquivo. Isso permite que os dados sejam lidos e gravados em qualquer local do arquivo.

- Usamos o método `seek` para mover o ponteiro de arquivo.

- `void seek(long pos)`
  - O argumento `pos` é o número do byte para o qual você deseja mover o ponteiro de arquivo.
  - Lança uma `IOException` se `pos` for menor que 0 ou se uma erro de I/O ocorrer.
  - A função permite que `pos` aponte para depois do final do arquivo. Ela não lança exceção neste caso.

# Exemplo — Sistema de Oficina de Carros



- Voltando ao exemplo da oficina de carros, gostaríamos de mudar o sistema para que pudéssemos ter acesso aleatório aos registros do arquivo.

- No sistema do exemplo, um objeto `Car` possui três atributos:
  - `registration`: uma String contendo o número de registro
  - `make`: uma String com o nome do fabricante
  - `price` um double contendo o preço

- Sabemos que `price` tem sempre 8 bytes. O problema são os outros dois campos, que podem variar de tamanho.
  - Para resolver esse problema, limitaremos cada um dos dois campos restantes a 10 caracteres apenas.
  - Com isso, cada variável do tipo String terá um byte para cada caractere mais dois bytes extra que guardarão um inteiro representando o tamanho da String.

# Exemplo — Sistema de Oficina de Carros



- Assim, o número máximo de bytes necessários para armazenar um registro no arquivo é calculado como:

| | |
|---|---|
| registration (String) | 12 bytes |
| make (String) | 12 bytes |
| price(double) | 8 bytes |
| **Total** | **32 bytes** |

# Exemplo — Sistema de Oficina de Carros



- Assim, o número máximo de bytes necessários para armazenar um registro no arquivo é calculado como:

| | |
|---|---|
| registration (String) | 12 bytes |
| make (String) | 12 bytes |
| price(double) | 8 bytes |
| **Total** | **32 bytes** |

- Ainda resta um problema: E se o número de um dos atributos String for menor que 10?
  - A melhor forma de lidar com isso é completar a string com espaços em branco, de modo que todo atributo String tenha exatamente 10 caracteres.

# Exemplo — Sistema de Oficina de Carros



- Assim, o número máximo de bytes necessários para armazenar um registro no arquivo é calculado como:

| | |
|---|---|
| registration (String) | 12 bytes |
| make (String) | 12 bytes |
| price(double) | 8 bytes |
| **Total** | **32 bytes** |

- Ainda resta um problema: E se o número de um dos atributos String for menor que 10?
  - A melhor forma de lidar com isso é completar a string com espaços em branco, de modo que todo atributo String tenha exatamente 10 caracteres.

- Isso implica que o tamanho de todo registro no arquivo sempre terá exatamente 32 bytes!

# Exemplo — Sistema de Oficina de Carros

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Analisar ProjetoRandomAccessCar

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

# FIM