UNIDADE 9 - PERSISTÊNCIA

Programação Orientada a Objetos

Prof. Marcel Hugo

FURB
UNIVERSIDADE DE BLUMENAU

1

Objetivos

- Ter visão geral do pacote java.nio.file Java SE 8
- Diferenciar arquivo binário de arquivo de texto
- Conceituar *stream*
- Salvar dados em arquivo
- Ler dados de arquivos
- Serializar (e desserializar) objetos

2

NIO.2

java.nio.file JAVA SE 8

FURB
UNIVERSIDADE DE BLUMENAU

3

API de Arquivos

- A NIO.2 (java.nio.file) é uma nova API de manipulação de arquivos que surgiu juntamente com o Java 7.
- Essa API define interfaces e classes para a JVM acessar arquivos, atributos de arquivos e o sistema de arquivos.
- Traz melhorias em relação a sua antecessora, fornecendo novos recursos, resolvendo antigos problemas e limitações e permitindo uma codificação mais limpa por meio de métodos mais intuitivos.
- Provê interoperabilidade com a classe *java.io.File,* principal classe da package *java.io* até SE6.

4

4

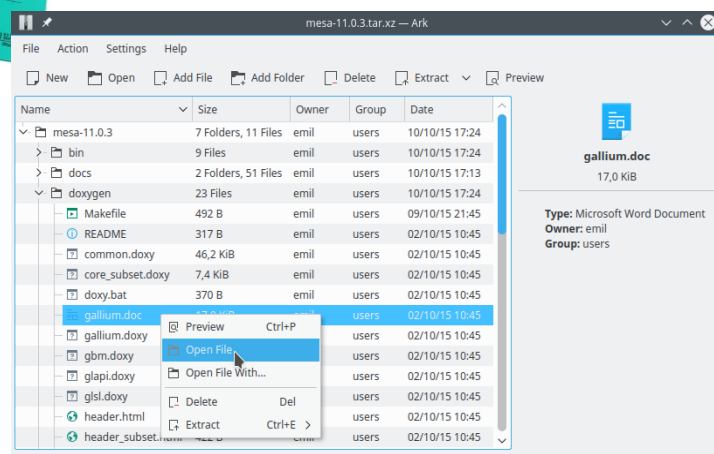
API de Arquivos

- Essa API fornece:
  - Acesso e manipulação de arquivos e diretórios do sistema de arquivos;
  - Navegação em árvores de diretórios;
  - Manipulação de atributos de arquivos e diretórios;
  - Acesso assíncrono para sockets e arquivos;
  - Melhorias nos sistemas de *buffers* de I/O;
  - Operações simples para mover, copiar e apagar arquivos;
  - Suporte a links simbólicos;
  - Capacidade de lidar com todos os sistemas de arquivo de forma unificada.

5

java.nio.file

Opera com o arquivo e diretórios , sem importar seu conteúdo.
Operações básicas, como mover, apagar, atributos, ...



6

Principais Pacotes e Subpacotes

- **java.nio.file:** pacote principal. Nele que se encontra a maioria das classes para a tarefa de manipulação de arquivos;
- **java.nio.file.attribute:** disponibiliza classes e interfaces para dar suporte à manipulação dos atributos de arquivos, como por exemplo, data de criação, última modificação, etc.;
- **java.nio.file.spi:** *Service Provider Implementors* - disponibiliza classes e interfaces que podem ser implementadas ou estendidas para dar suporte a novos sistema de arquivos ou serviços específicos.

7

7

Principais Classes e Interfaces

- **Interface Path**
- **Classe Paths**
- **Classe Files**
- **Classe abstrata FileSystem**
- **Classe FileSystems**

- Documentação:

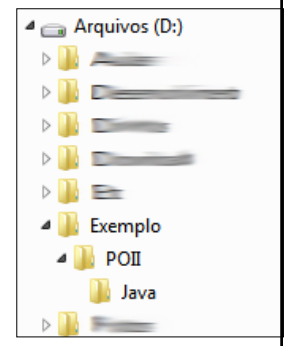
http://docs.oracle.com/javase/8/docs/api/java/nio/file/package-summary.html

8

8

Diretórios

- Caractere separador:
  - barra (/) (Unix)
  - barra invertida (\) (Windows)
- Caminho absoluto
  - Contém todos os diretórios desde o raiz
    - Exemplo: D:\Exemplo\POII\Java\Prog.java
- Caminho relativo
  - O caminho é relativo ao diretório atual
    - Java\Prog.java



9

Interface Path

- Representa um caminho para um diretório ou arquivo.
- Principais métodos:

int compareTo(Path other): compara dois paths lexicograficamente.
Path getFileName() : retorna o nome do arquivo ou diretório do path.
Path getParent(): retorna o path hierarquicamente superior.
int getNameCount(): retorna a quantidade de elementos do path.
Path getName(int index): retorna o elemento indicado por index.
boolean startsWith(Path other): indica se o path começa com o path indicado.
boolean endsWith(Path other): indica se o path termina com o path indicado.
boolean isAbsolute(): indica se o path é absoluto ou não.
Path toAbsolutePath(): retorna uma versão absoluta do path.
String toString(): retorna uma string equivalente ao path.
FileSystem getFileSystem(): retorna o FileSystem que criou o path.

10

Classe Paths

- Substitui a classe **File** do pacote **java.io**
- *Utility class*
- Principais métodos:

static Path get(String... path): retorna um objeto Path indicado pela string.
static Path get(URI uri): retorna um objeto Path indicado pela URI.

11

11

Path e Paths

```
Path arq1 = Paths.get("\\arquivos\\teste.txt");
Path arq2 = Paths.get("\\arquivos", "teste.txt");

System.out.format("%s é igual a %s? %b%n", arq1, arq2,
arq1.compareTo(arq2));
System.out.format("Nome do arquivo: %s%n", arq1.getFileName());
System.out.format("Diretório do arquivo: %s%n", arq1.getParent());
System.out.format("String do path: %s%n", arq1.toString());
System.out.format("Caminho absoluto: %s%n", arq1.toAbsolutePath());
```

12

12

Classe Files

- Executa operações em arquivos e diretórios;
- *Utility class*
- Métodos principais (Obs: todos lançam IOException):

static Path copy(Path source, Path target, CopyOption... options)
static Path move(Path source, Path target, CopyOption... options)
static Path createFile(Path path, FileAttribute<?>... attrs)
static void Path delete(Path path) ou static boolean deleteIfExists(Path path)
static boolean exists(Path path, LinkOption… options)
static boolean isDirectory(Path path, LinkOption… options)
static boolean isExecutable(Path path)
static boolean isSameFile(Path path, Path path2)
static List<String> readAllLines()

13

13

Classe Files

```
Path path1 = Paths.get("C:\\arquivos\\Teste.txt");
Path path2 = Paths.get("C:\\arquivos\\Teste2.txt");
try {
        Path file = Files.createFile(path1);
        Files.copy(file, path2);
}
catch (IOException e) {
        e.printStackTrace();
}
```

14

14

Classes FileSystem e FileSystems

- Classe abstrata FileSystem
  - Representa o sistema de arquivos do computador;
  - O método *getPath()* é uma fábrica para objetos Path;
  - O método *getSeparator()* retorna a String do separador usado pelo sistema operacional.
- Classe FileSystems
  - Estende FileSystem;
  - O método *static getDefault()* fornece acesso ao sistema de arquivos padrão.

```
FileSystem fileSystem = FileSystems.getDefault();
Path arq4 = fileSystem.getPath("C:\\arquivos\\teste.txt");
```

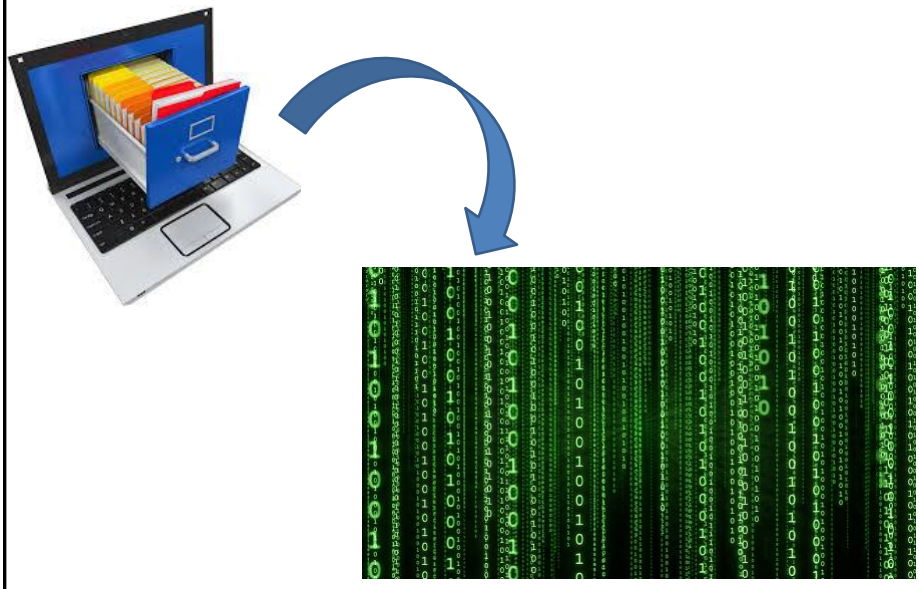
15

15

PERSISTÊNCIA DE DADOS

FURB
UNIVERSIDADE DE BLUMENAU

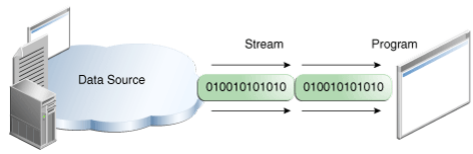
16



17

I/O Stream

- Um *Stream* (ou **fluxo** sequencial de bytes) é um objeto que ou entrega dados para seu destinatário (tela, arquivo, etc), ou obtém dados de uma fonte (teclado, arquivo, socket, etc).
- Atuar como um **túnel** entre a fonte de dados e o destino.



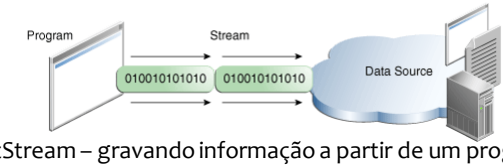
InputStream - lendo informação para um programa.

- Fonte / destino de dados: qualquer coisa que armazene, gere ou consuma dados.
- Exemplos: arquivos em disco, outros programas, dispositivos periféricos, socket de rede, etc

18

I/O Stream

- Fluxos básicos, leitores e gravadores conseguem processar apenas caracteres ou bytes individuais. Você precisa combiná-los com outras classes para processar linhas ou objetos inteiros.



OutputStream – gravando informação a partir de um programa.

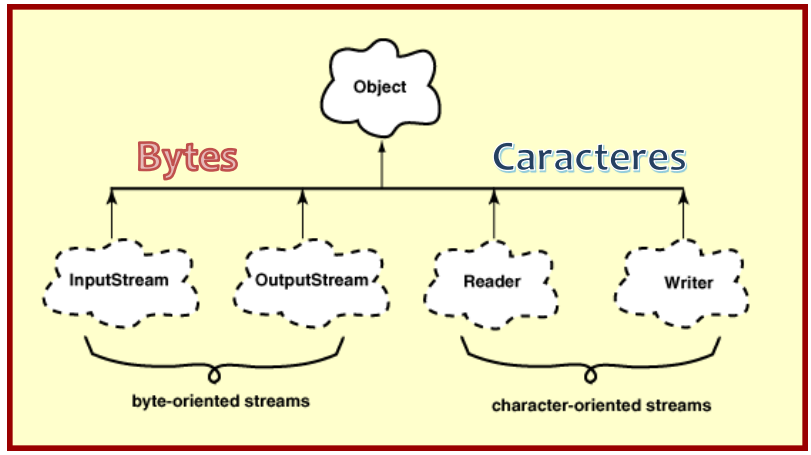
19

Arquivo Texto X Arquivo Binário

- Arquivo texto
  - Os bits representam caracteres
  - Podem ser lidos por editores de texto
  - São "legíveis" para os humanos
- Arquivos binários
  - Os bits representam dados
  - Utilizam qualquer sequência de bytes
  - Mais eficiente de processar
- Exemplo: para gravar o inteiro 12345
  - Texto: 5 caracteres: '1' '2' '3' '4' '5'
  - Binário: um *int* (quatro bytes): 0 0 48 57 (48x256 + 57)

20

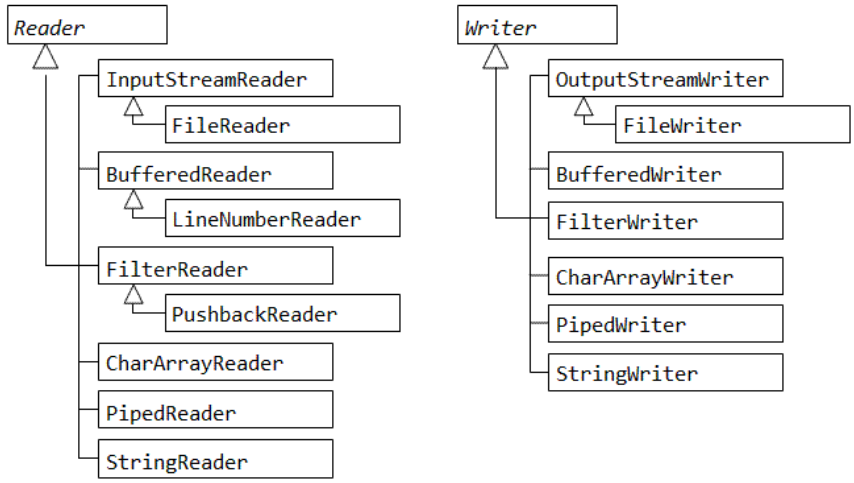
Classes básicas de arquivos



21

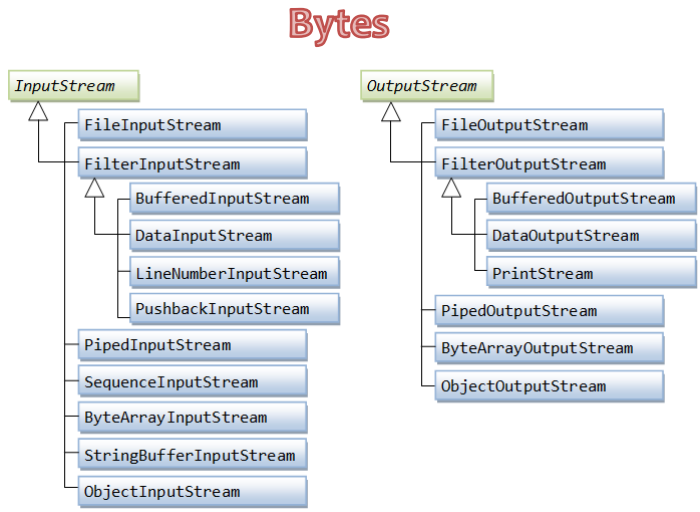
Hierarquia de classes de *caracter*

Caracteres



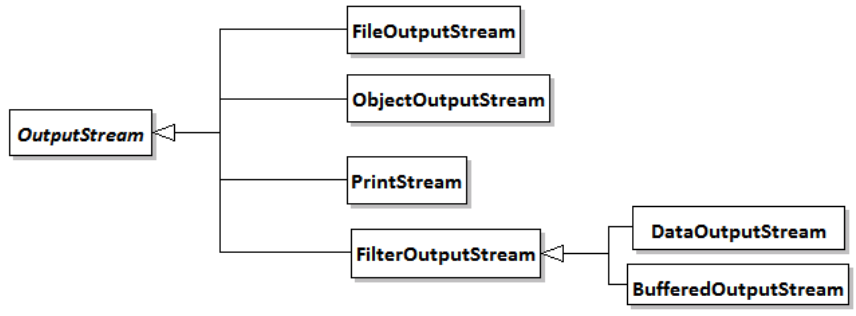
22

Hierarquia de classes de *streams*



23

Gravação de arquivos binários



24

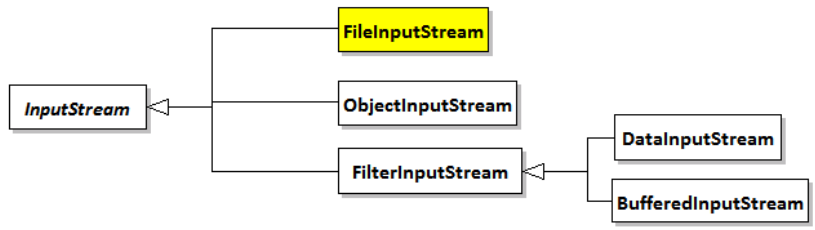
Gravação de arquivos binários

- **FileOutputStream** grava arquivos em formato binário

Método	Descrição
`FileOutputStream(String)` `FileOutputStream(File)`	Cria o arquivo especificado
`FileOutputStream(String, boolean)` `FileOutputStream(File, boolean)`	Permite abrir o arquivo para alteração
`write(int)`	Escreve um byte no arquivo
`write(byte[])` `write(byte[], int, int)`	Escreve um vetor de bytes no arquivo
`close()`	Fecha o arquivo

25

Leitura de arquivos binários



26

Leitura de arquivos binários

- **FileInputStream** lê arquivos em formato binário

Método	Descrição
`FileInputStream(File)` `FileInputStream(String)`	Abre um arquivo
`int read()`	Lê um byte
`int read(byte[])` `int read(byte[], int, int)`	Lê um vetor de bytes
`close()`	Fecha o arquivo

27

Exceções

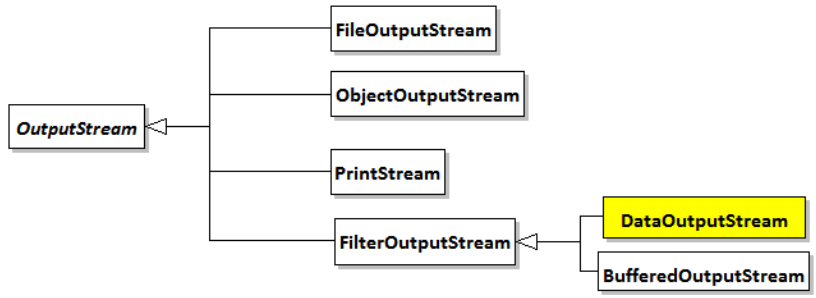
- Exceções mais comuns lançadas pelas classes de manipulação de arquivos

Exceção	Descrição
`SecurityException`	Usuário sem permissão para gravar dados no arquivo
`FileNotFoundException`	Arquivo não existe
`IOException`	Falha de leitura/gravação

28

Manipulação de dados primitivos e Strings

- A classe DataOutputStream permite gravar dados primitivos e Strings num *stream* de saída



29

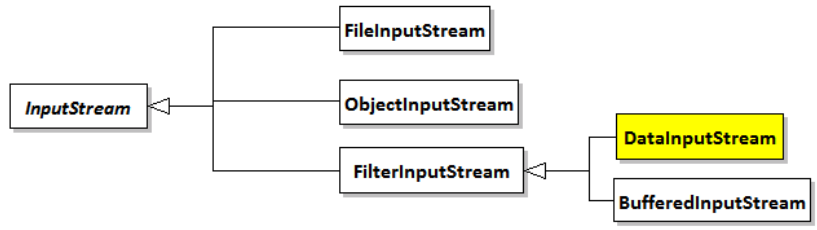
Métodos do DataOutputStream

Método	Descrição
`DataOutputStream(OutputStream)`	Cria um DataOuputStream, direcionando os dados para OutputStream
`writeByte(int)`	Grava um byte
`writeChar(char)`	Grava um caractere
`writeFloat(float)`	Grava um valor float
`writeDouble(double)`	Grava um valor double
`writeInt(int)`	Grava um valor int
`writeLong(long)`	Grava um valor long
`writeUTF(String)`	Grava uma String

30

DataInputStream

- Classe usada para obter dados primitivos ou Strings



31

Métodos do DataInputStream

Método	Descrição
DataInputStream(InputStream)	Cria um DataInputStream obtendo os dados de InputStream
byte readByte()	Lê um byte
char readChar()	Lê um caractere
float readFloat()	Lê um valor float
double readDouble()	Lê um valor double
int readInt()	Lê um valor int
long readLong()	Lê um valor long
string readUTF()	Lê uma string

32

Observações

- DataInputStream e DataOutputStream leem e gravam tipos primitivos Java e Strings de forma independente de máquina
- Deve-se ler os dados na mesma ordem em que eles foram gravados.
- A exceção EOFException pode ser utilizada para detectar se o arquivo ainda possui dados

33

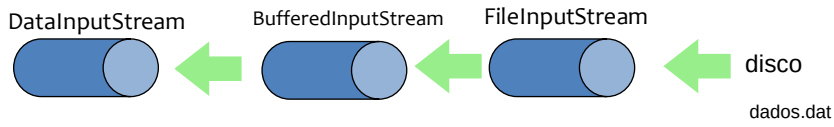
BufferedInputStream e BufferedOutputStream

- Usados para aumentar a velocidade de leitura e gravação reduzindo o número de leituras e gravações no dispositivo.
-

Método	Descrição
BufferedOutputStream(OutputStream)	Cria um Buffer para OutputStream
BufferedOutputStream(OutputStream, int)	Cria um Buffer para OutputStream, definindo o tamanho do buffer
flush()	"descarrega" o buffer no outputStream

34

Exemplo:




```
File arquivo = new File("D:\\dados.dat");

FileInputStream fs = new FileInputStream(arquivo);
BufferedInputStream bs = new BufferedInputStream(fs);
DataInputStream ds = new DataInputStream(bs);
```

- Neste exemplo, deve-se utilizar os métodos do DataInputStream para ler os dados do stream

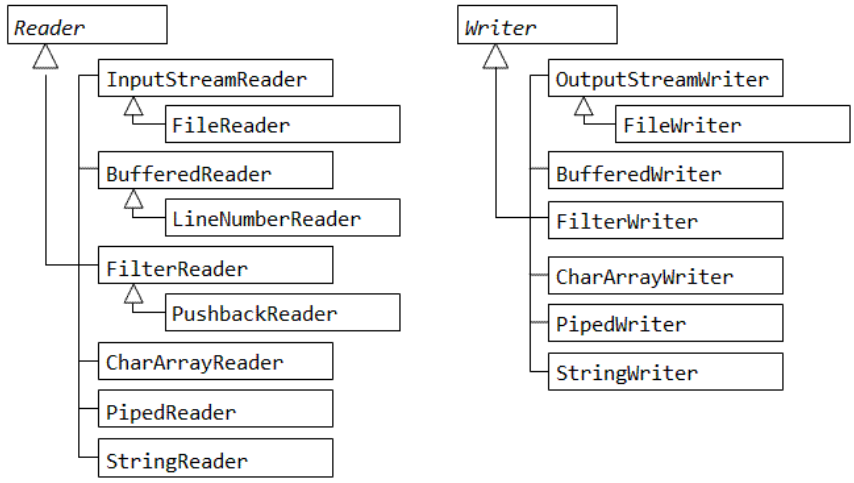
35

Readers e Writers

- São semelhantes aos InputStream e OutputStream, porém trabalham com caracteres ao invés de bytes.
- Por causa disso, são mais adequados para arquivos de texto.
- Contudo, eles realizam uma conversão de bytes para char durante a leitura/escrita do arquivo, fazendo com que sejam mais lentos do que os *streams*.

36

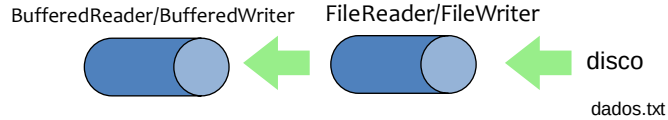
Readers e Writers



37

BufferedReader e BufferedWriter

- Leitor e escritor de arquivos com buffers, semelhantes ao BufferedInputStream e BufferedOutputStream.




```
Path path = Paths.get("C:\\arquivos\\teste.txt");

FileWriter fw = new FileWriter(path.toFile()));
BufferedWriter writer = new BufferedWriter( fw );
```

38

BufferedReader e BufferedWriter

- Métodos:

Método	Descrição
BufferedReader(FileReader)	Cria um Buffer para FileReader
BufferedWriter(FileWriter)	Cria um Buffer para FileWriter
flush()	"descarrega" o buffer
close()	Fecha o leitor/escritor
String readLine()	O leitor lê uma linha do arquivo.
write(String)	O escritor escreve uma String.

39

39

Serialização

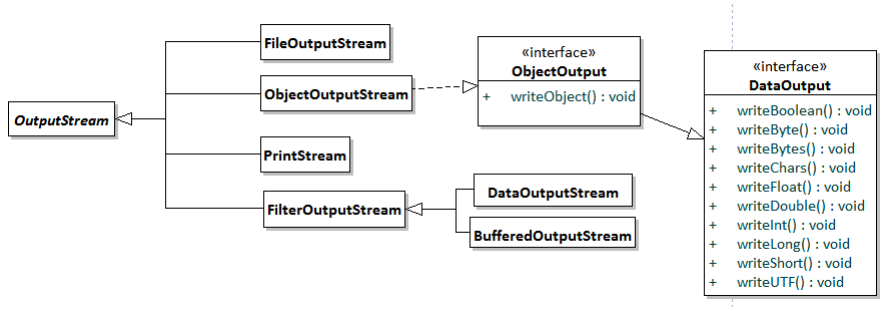
- Serializar um objeto é transformá-lo em bytes, de modo que poderá ser armazenado em disco ou transmitido por um *stream*.
- A serialização de objeto em Java é feita por meio das classes **ObjectOutputStream** e **ObjectInputStream**.
- Somente objetos de classes que implementam a interface **Serializable** podem ser serializados.

```
class MyClass implements Serializable {
  // atributos
  public String toString() { ...}
}
```

40

40

Serialização - ObjectOutputStream



41

ObjectOutputStream e ObjectInputStream

- ObjectOutputStream
  - Utilizado para serializar um objeto
  - Realiza a interface *ObjectOutput* que por sua vez estende a interface *DataOutput*
- ObjectInputStream
  - Utilizado para desserializar um objeto
  - Realiza a interface *ObjectInput* que por sua vez estende a interface *DataInput*

42

Exceções lançadas

- ClassNotFoundException
  - Ocorre quando se tenta desserializar objeto de uma classe desconhecida
- NotSerializableException.
  - Ocorre quando se quer serializar um objeto de uma classe que não implementa a interface *Serializable*

43

Observações

- Podem ser gravados vários objetos no stream.
- Para ler os objetos do stream, deve-se executar as operações de leitura na mesma ordem em que foram executadas as operações de gravação
- Para um objeto ser serializado, este deve implementar a interface *Serializable*
- Valores de variáveis estáticas não são serializados.
- Para não serializar determinada variável de instância, introduzir o modificador *transient*, como abaixo:
  - `private transient String campo;`

44