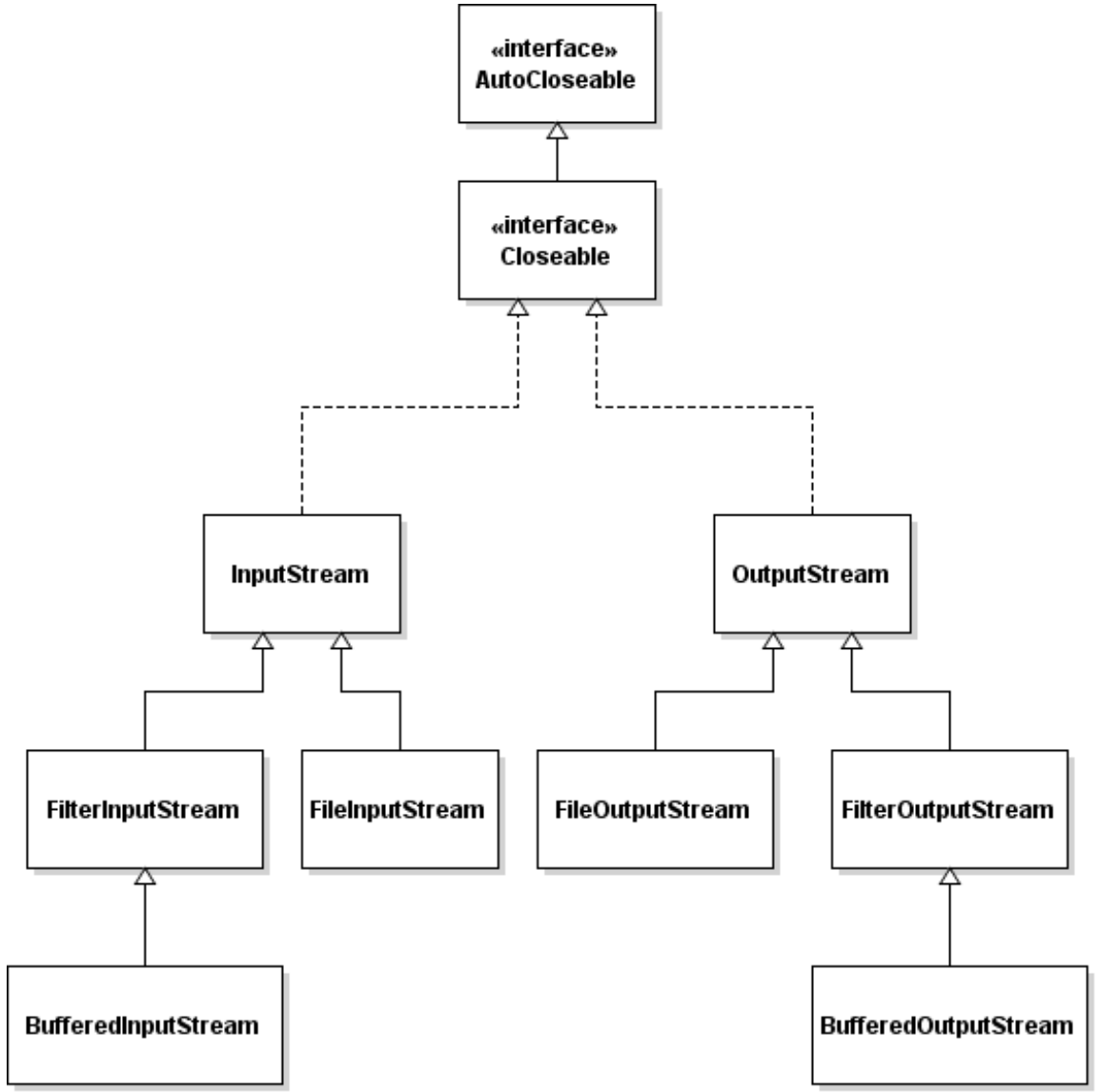
APLICAÇÃO DA LINGUAGEM DE PROGRAMAÇÃO ORIENTADA À OBJETOS – ALPOO

Aula – Java File I/O

Prof. Danilo Pereira - danilo.pereira1@docente.unip.br

Read and Write Text File in Java - Introduction



https://www.codejava.net/java-se/file-io/how-to-read-and-write-binary-fi
java

How to Read and Write Text File in Java

In this classe, we'll see you **how to read from and write to text (or character) files using classes available in the java.io package**.

First, let's look at the different classes that are capable of reading and writing character streams.

Reader, InputStreamReader, FileReader and

Reader is the **abstract class** for reading character streams. It implements the following fundamental methods:

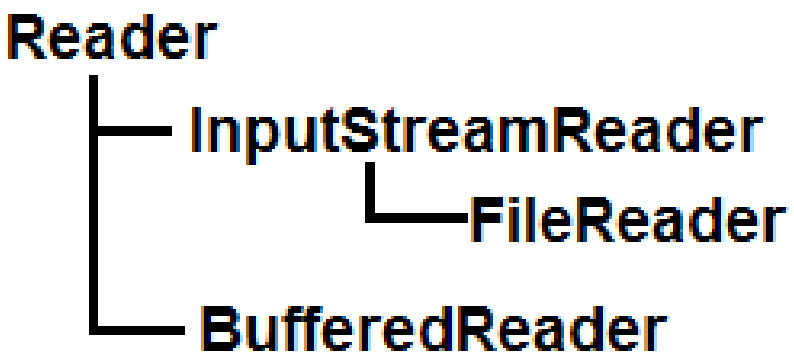
read(): reads a single character.
read(char[]): reads an array of characters.
skip(long): skips some characters.
close(): closes the stream.

InputStreamReader is a bridge from byte streams to character streams. It converts bytes into characters using a specified charset. The charset can be default character encoding of the operating system, or can be specified explicitly when creating an InputStreamReader.

Reader, InputStreamReader, FileReader and BufferedReader

FileReader is a **convenient class** for reading text files using the default character encoding of the operating system.

BufferedReader reads text from a character stream with efficiency (characters are buffered to avoid frequently reading from the underlying stream) and provides a convenient method for reading a line of text readLine().



Writer, OutputStreamWriter, FileWriter and BufferedWriter

Writer is the abstract class for writing character streams. It implements the following fundamental methods:

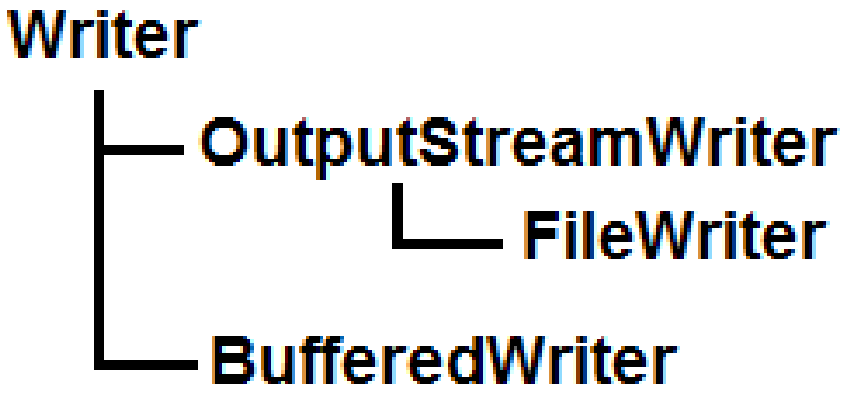
write(int): writes a single character.
write(char[]): writes an array of characters.
write(String): writes a string.
close(): closes the stream.

OutputStreamWriter is a bridge from byte streams to character streams.
Characters are encoded into bytes using a specified charset.
The charset can be default character encoding of the operating system, or can be specified explicitly when creating an OutputStreamWriter.

Writer, OutputStreamWriter, FileWriter and BufferedWriter

FileWriter is a convenient class for writing text files using the default character encoding of the operating system.

BufferedWriter writes text to a character stream with efficiency (characters, arrays and strings are buffered to avoid frequently writing to the underlying stream) and provides a convenient method for writing a line separator: newLine().



Character Encoding and Charset

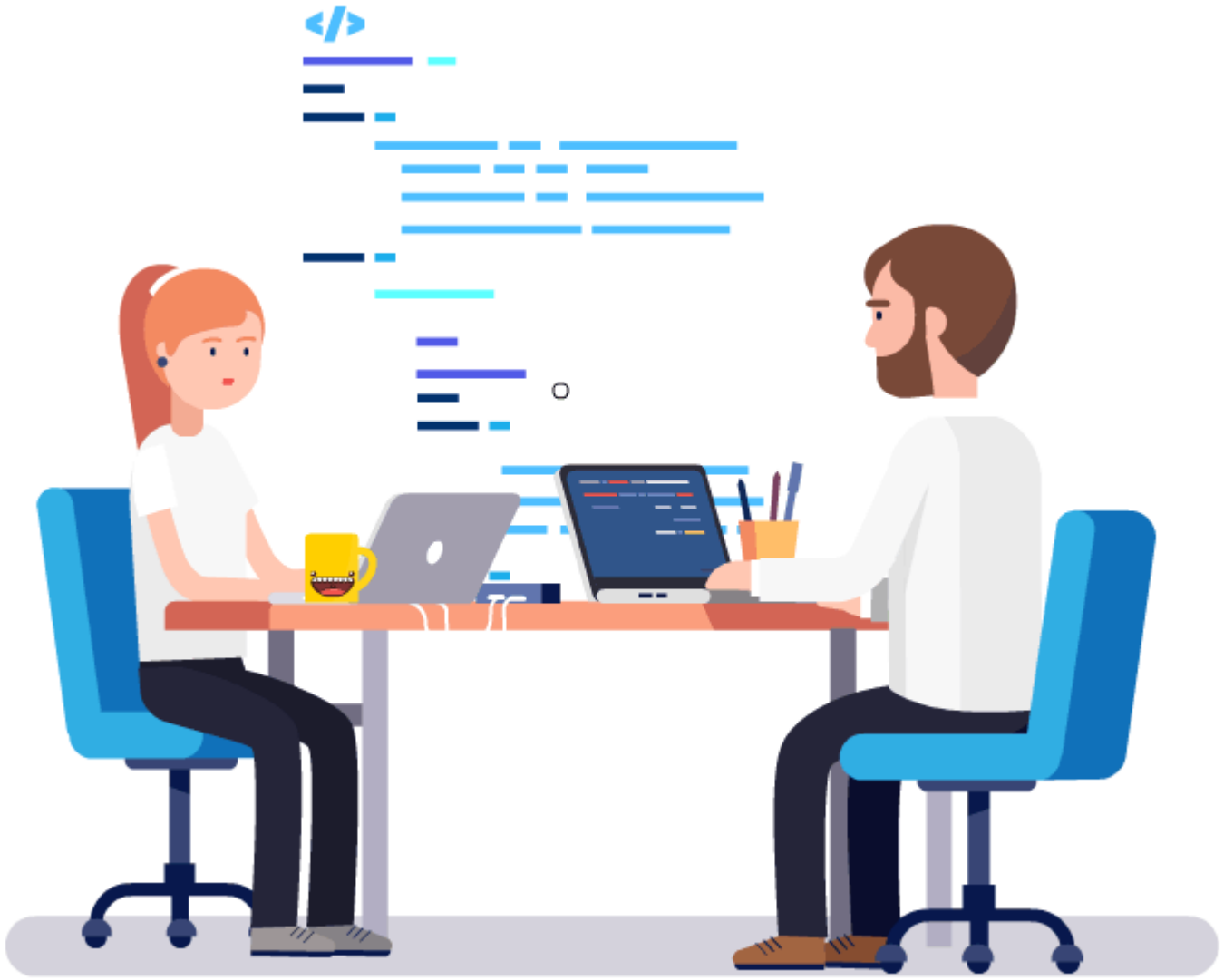
When constructing a reader or writer object, the **default character encoding of the operating system** is used (e.g. Cp1252 on Windows):

```
FileReader reader = new FileReader("C:\\MyFile.txt");
FileWriter writer = new FileWriter(" C:\\ YourFile.txt");

InputStreamReader reader = new InputStreamReader(new
FileInputStream(" C:\\MyFile.txt"), "UTF-16");

OutputStreamWriter writer = new OutputStreamWriter(new
FileOutputStream(" C:\\YourFile.txt"), "UTF-8");
```

VAMOS PRATICAR ?



Java Writing from Text File Example

```
import java.io.FileWriter;
import java.io.IOException;

public class TextFileWritingExample1 {

    public static void main(String[] args) {
        try {
            FileWriter writer = new
FileWriter("MyFile.txt", true);
            writer.write("Hello World");
            writer.write("\r\n");   // write new line
            writer.write("Good Bye!");
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class TextFileWritingExample2 {
    public static void main(String[] args) {
        try {
            FileWriter writer = new FileWriter("MyFile.txt",
true);
            BufferedWriter bufferedWriter = new
BufferedWriter(writer);

            bufferedWriter.write("Hello World");
            bufferedWriter.newLine();
            bufferedWriter.write("See You Again!");

            bufferedWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Java Reading from Text File Example

```
import java.io.FileReader;
import java.io.IOException;

public class TextFileReadingExample1 {
    public static void main(String[] args) {
        try {
            FileReader reader = new
FileReader("MyFile.txt");
            int character;

            while ((character = reader.read()) != -1)
{
                System.out.print((char) character);
            }
            reader.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;

public class TextFileReadingExample2 {
    public static void main(String[] args) {
        try {
            FileInputStream inputStream = new
FileInputStream("MyFile.txt");
            InputStreamReader reader = new
InputStreamReader
            (inputStream, "UTF-16");
            int character;

            while ((character = reader.read()) != -1) {
                System.out.print((char) character);
            }
            reader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

How to read text file line by line in Java

In Java File I/O programming, the classes **BufferedReader** and **LineNumberReader** allows reading a sequence of lines from a text file, in a line-by-line fashion, by using their **readLine() method**.

The LineNumberReader is a subclass of the BufferedReader class. The only difference is that, the LineNumberReader class keeps track of the current line number, whereas the BufferedReader class does not.

Reading all lines from a text file using the BufferedReader class is as easy as follows:

How to read text file line by line in Java – Example 1

```
String filePath = "Path/To/Your/Text/File.txt";

try {
        BufferedReader lineReader = new BufferedReader(new
FileReader(filePath));
    String lineText = null;

    while ((lineText = lineReader.readLine()) != null) {
        System.out.println(lineText);
    }

    lineReader.close();
} catch (IOException ex) {
    System.err.println(ex);
}
```

How to read text file line by line in Java – Example 2

```
String filePath = "Path/To/Your/Text/File.txt";

try {
BufferedReader lineReader = new BufferedReader(new
FileReader(filePath));
String lineText = null;

ArrayList<String> listLines = new ArrayList<String>();

while ((lineText = lineReader.readLine()) != null) {
    listLines.add(lineText);
}

lineReader.close();
} catch (IOException ex) {
    System.err.println(ex);
}
```

How to read text file line by line in Java – Example 3

```
String filePath = "Path/To/Your/Text/File.txt";

try {
    LineNumberReader lineReader = new LineNumberReader(new
FileReader(filePath));
    String lineText = null;

    while ((lineText = lineReader.readLine()) != null) {
        System.out.println(lineReader.getLineNumber() + ": " + lineText);
    }

    lineReader.close();
} catch (IOException ex) {
    System.err.println(ex);
}
```

How to read text file line by line in Java – Example 4

The LineNumberReader class provides the getLineNumber() method which returns the current line number, so we can use it for checking conditions on line numbers, for example:

Printing only the even lines

```
while ((lineText = lineReader.readLine()) != null) {
    int lineNumber = lineReader.getLineNumber();
    if (lineNumber % 2 == 0) {
        System.out.println(lineNumber + ": " + lineText);
    }
}
```

How to read text file line by line in Java – Example 5

Printing only the lines in a specific range

```
while ((lineText = lineReader.readLine()) != null) {
    int lineNumber = lineReader.getLineNumber();
    if (lineNumber >= 50 && lineNumber <= 100) {
        System.out.println(lineNumber + ": " + lineText);
    }
}
```

EXERCICIO DE FIXAÇÃO

Objetivo: Desenvolver um tela (Java Swing) para cadastro de clientes

A tela deve conter os seguintes campos:

- Nome,CPF, RG: idade, email e telefone
- Endereço: rua, numero, bairro, CEP, cidade e estado

Regras de negócio

- **Botão Salvar -> Salvar as informações num arquivo de texto**
- **Botão Limpar ->** Limpar todos os campos da tela.
- **Validação:** Nome, CPF, email e telefone são obrigatórios

Exemplo – Escrita em arquivo de texto

```
FileWriter writer = new FileWriter("C://Teste.txt", true);
BufferedWriter bufferedWriter = new
BufferedWriter(writer);

StringBuffer texto = new StringBuffer();
texto.append("Hello World \n");
texto.append("Danilo R. Pereira! \n");
texto.append("See You Again! \n");

bufferedWriter.write(texto.toString());

bufferedWriter.close();
```

Arquivos e diretorios

Nessa aula, vamos aprender como escrever código para listar o conteúdo (arquivos) de um diretório.

Em Java, para consultar uma lista de arquivos em um diretório específico, usamos a classe **File no pacote java.io** e seguimos estas etapas:

1) Instanciando a classe File

File dir = **new File**(“C:/Path/To/My/Directory”);
or
File dir = **new File**(“C:/Path/To/My/MyFile”);

Arquivos e diretorios (cont.)

2) Usar os métodos da classe File

- ***String[] list()***
- ***String[] list(FilenameFilter filter)***
- ***File[] listFiles()***
- ***File[] listFiles(FileFilter filter)***
- ***File[] list(FilenameFilter filter)***

Os métodos list () retornam uma **matriz de Strings**.
Os métodos listFiles () retornam uma **matriz de objetos File**.
Os métodos sem argumento **listam tudo no diretório.**
Os métodos baseados em argumento listam apenas arquivos e diretórios que satisfaçam o filtro fornecido.

Exemplo 1 – Listando os arquivos de um diretorio

```
String dirPath = "C:/Path/To/My/Directory/";

File dir = new File(dirPath);

String[] files = dir.list();

if (files != null && files.length == 0) {
    System.out.println("O diretório está vazio !!!");
} else {
    for (String aFile : files) {
        System.out.println(aFile);
    }
}
```

Exemplo 2 – Listando os arquivos de um diretorio

```
String dirPath = "C:/Path/To/My/Directory/";

File dir = new File(dirPath);

File[] files = dir.listFiles();
if (files != null && files.length == 0) {
    System.out.println("O diretório está vazio !!!");
} else {
    for (File aFile : files) {
        System.out.println(aFile.getName() + " - " +
aFile.length());
    }
}
```

Interface FilenameFilter

1) Usando a interface FilenameFilter para criar o critério de busca

```
// Filtrando apenas arquivo com a extensao .txt
FilenameFilter txtFilter = new
FilenameFilter() {
        public boolean accept(File file, String
name) {
                if (name.endsWith(".txt")) {
                    return true;
                } else {
                    return false;
                }
                }
```

Interface FileFilter

1) Usando a interface FileFilter para criar o critério de busca

```
// Filtrando apenas arquivo com mais que 3BM
FileFilter sizeFilter = new FileFilter() {
        public boolean accept(File file) {
            if (file.isFile() && file.length() > 3 * 1024
* 1024) {
                return true;
            } else {
                return false;
            }
        }
};
```

Exemplo 3 – Listando os arquivos de um diretorio (Filtrado)

```
String dirPath = "C:/Path/To/My/Directory/";

File dir = new File(dirPath);
File[] files = dir.listFiles(txtFilter);

if (files != null && files.length == 0) {
    System.out.println("O diretório está vazio !!!");
} else {
    for (File aFile : files) {
        System.out.println(aFile.getName() + " - " +
aFile.length());
    }
}
```

Exemplo 3 – Listando os arquivos de um diretorio (Filtro)

```
String dirPath = "C:/Path/To/My/Directory/";

File dir = new File(dirPath);
File[] files = dir.listFiles(sizeFilter);

if (files != null && files.length == 0) {
    System.out.println("O diretório está vazio !!!");
} else {
    for (File aFile : files) {
        System.out.println(aFile.getName() + " - " +
aFile.length());
    }
}
```

Copiando arquivos

```
 public static void copyFile(String filePath, String dir)
{
                Path sourceFile = Paths.get(filePath);
                Path targetDir = Paths.get(dir);
                Path targetFile =
targetDir.resolve(sourceFile.getFileName());

                try {
                    Files.copy(sourceFile, targetFile);
                } catch (FileAlreadyExistsException ex) {
                    System.err.format("Arquivo %s já existe.",
targetFile);
                } catch (IOException ex) {
                    System.err.format("I/O Error when copying
```

Renomeando arquivos

```
File sourceFile = new
File("<caminho>/<nome_arquivo>");
File destFile = new
File("<caminho>/<novo_nome_arquivo>");

if (sourceFile.renameTo(destFile)) {
   System.out.println(“Arquivo renomeado com
sucesso !!!")
} else {
    System.out.println(“Falha ao renomear o
arquivo !!!");
```

Deletando arquivos e diretorios

```
public static void removeDirectory(File dir) {
        if (dir.isDirectory()) {
            File[] files = dir.listFiles();
            if (files != null && files.length > 0) {
                for (File aFile : files) {
                    removeDirectory(aFile);
                }
            }
            dir.delete();
        } else {
            dir.delete();
        }
}
```

```
public static void cleanDirectory(File
dir) {
            if (dir.isDirectory()) {
                File[] files = dir.listFiles();
                if (files != null &&
files.length > 0) {
                    for (File aFile : files) {

removeDirectory(aFile);
                    }
                }
            }
}
```

EXERCÍCIO

1) Criei uma **tela** com as seguintes opções:
- a) Criar arquivo criptografado
  - Para isso, o usuário deve informar os seguintes campos:
    - Texto longo (conteúdo do arquivo)
    - Senha (chave de criptografia)
    - Nome do arquivo
    - Botão salvar
- b) Ler arquivo criptografado
  - Para isso, o usuário deve informar os seguintes campos:
    - Nome do arquivo
    - Senha (chave de criptografia)
    - Botão abrir
  - Deve ser exibido na tela:
    - Texto longo (conteúdo descriptografado do arquivo)

LINK UTEIS

- **Java File I/O (Reading & Writing)**
  - https://www.youtube.com/watch?v=hgF21imQ_Is

- **Java File Input/Output - It's Way Easier Than You Think**
  - https://www.youtube.com/watch?v=ScUJx4aWRi0

- **Java: Read a CSV File into an Array**
  - https://www.youtube.com/watch?v=-Aud0cDh-J8

- **Criptografia em Java**
  - https://www.devmedia.com.br/utilizando-criptografia-simetrica-em-java/31170