Plataforma Java

Java IO
Java NIO

Java IO

1. File
2. Stream
3. Byte Stream
4. Text Stream
5. Data Stream
6. Object Stream
7. Buffered Read/Write

Java NIO

13. Classe Object
14. Clonagem de Objetos
15. Classe String

Java IO

File

A classe File representa um arquivo e diretório

Define o caracter de separador de diretórios e arquivos, que é dependente de plataforma: File.separatorChar e File.separator

Métodos para criar e excluir arquivos e diretórios, verificar existências, tamanho, se é arquivo ou diretório, se é oculto, se é executável, renomear, obter metadados do arquivo, filtrar arquivos de um diretório, etc

Java IO

Stream

I/O Stream é uma “abstração” que representa uma fonte para entrada ou destino para saída

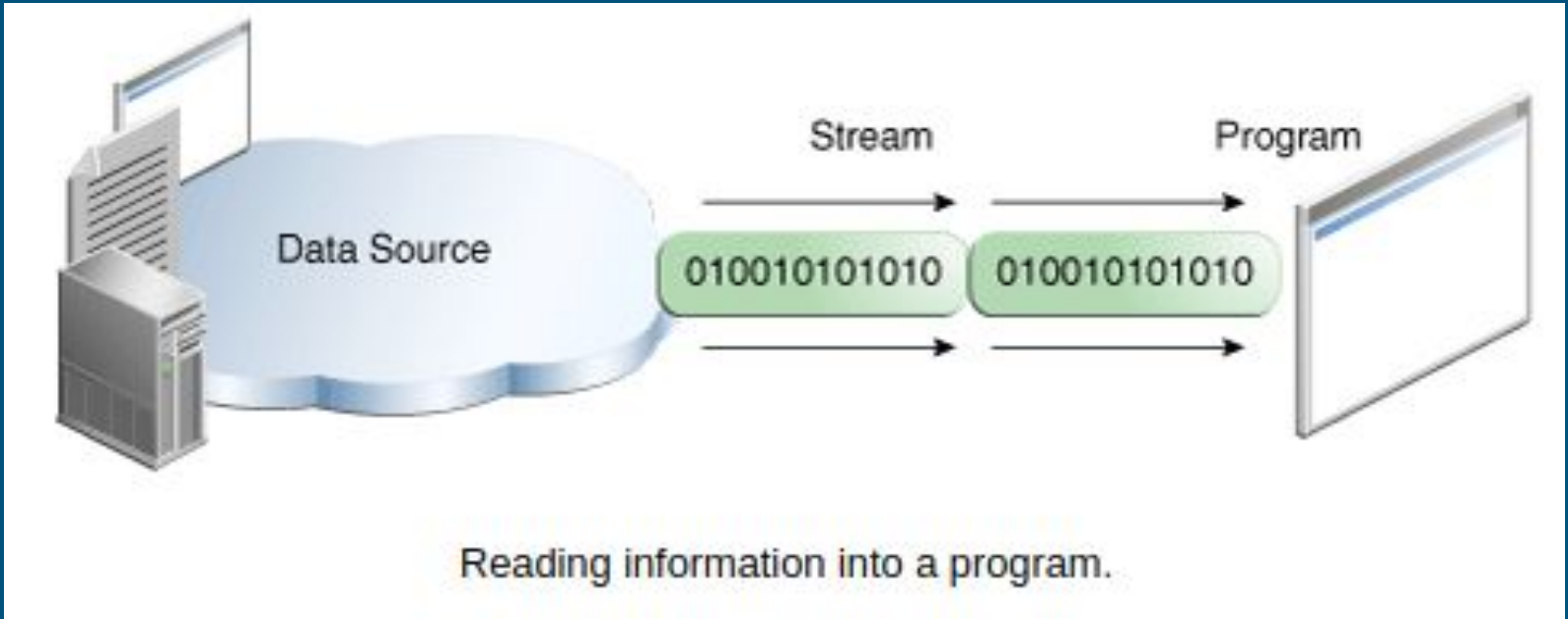
Pode representar diversos tipos de fontes e destinos, incluindo arquivos, dispositivos, outros programas e arrays na memória.

Streams suportam muitos tipos de dados, incluindo bytes, tipos primitivos (int, float, double, ...), caracteres localizados e objetos.

Não importa como trabalha internamente, todo stream representa o mesmo modelo de programação: Stream é uma sequencia de dados. O programa usa um *input stream* para ler dados e um *output stream* para escrever dados

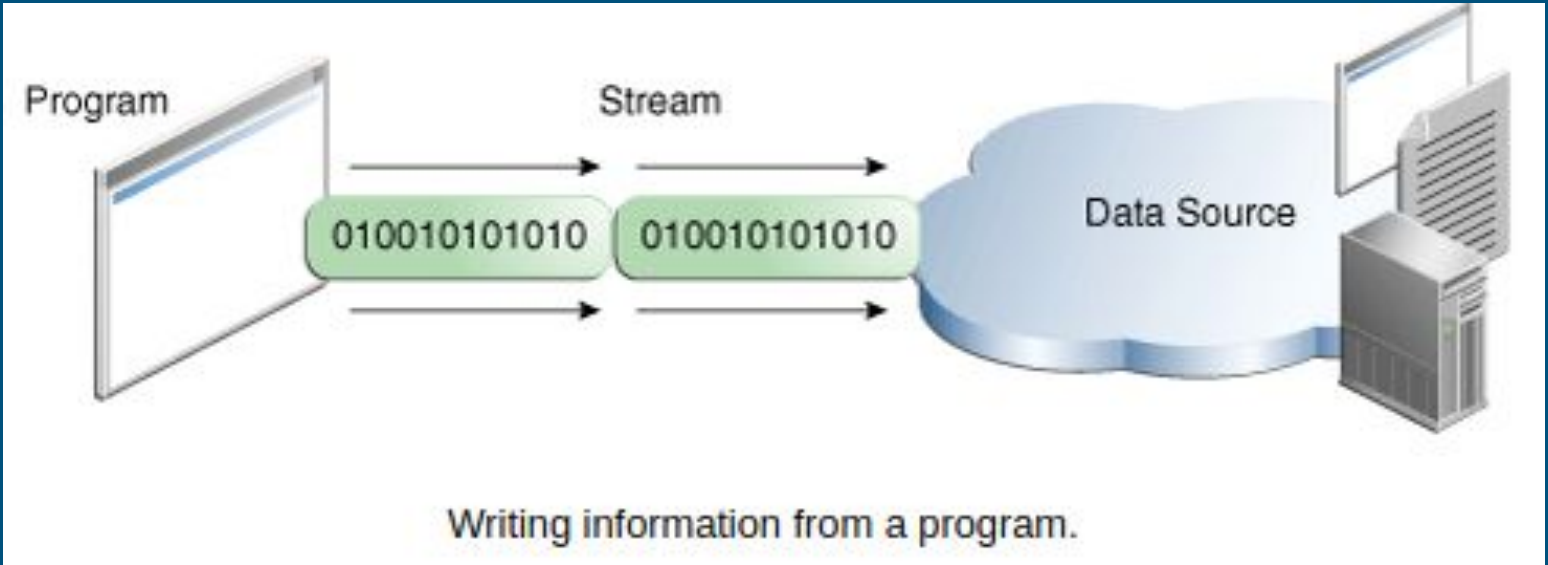
Java IO

Stream



Java IO

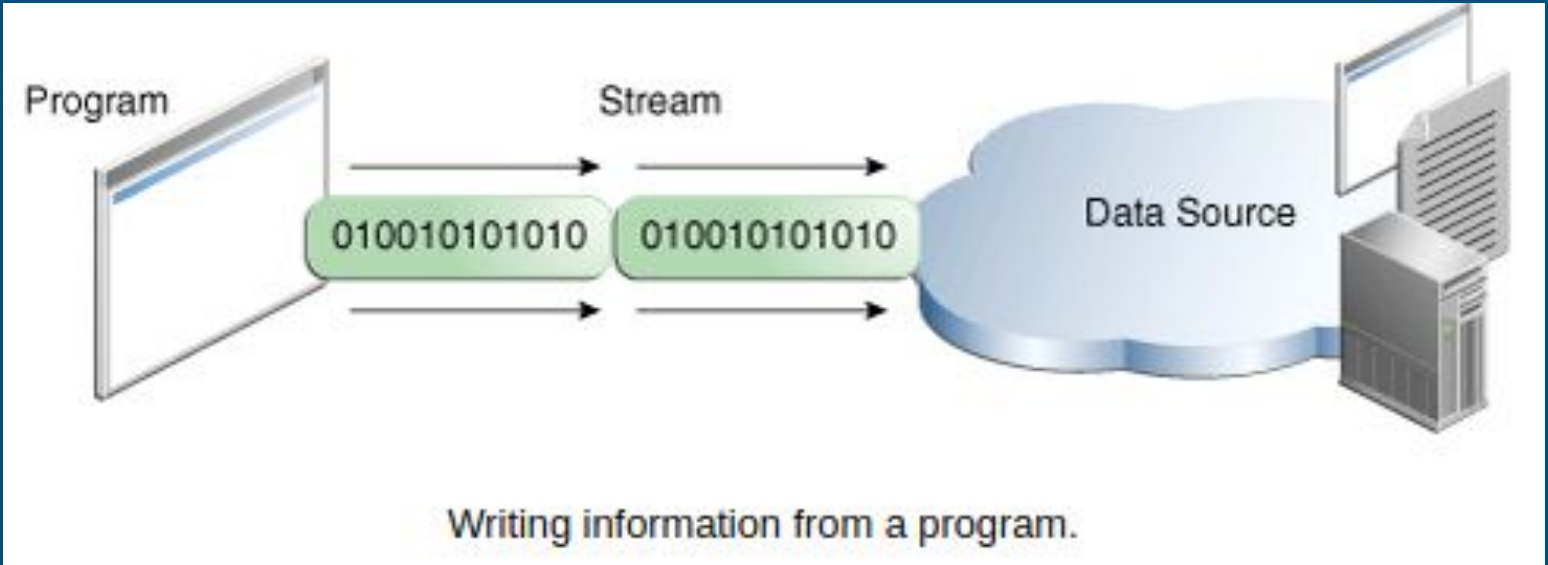
Stream



Writing information from a program.

Java IO

Stream



Writing information from a program.

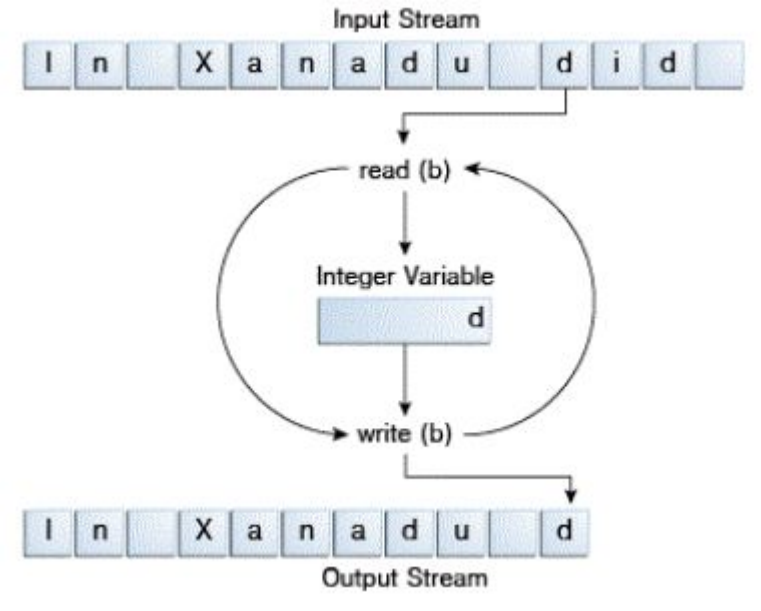
Java IO

Byte Stream

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

public class CopyBytes {
    public static void main(String[] args) throws IOException {

        FileInputStream in = null;
        FileOutputStream out = null;

        try {
            in = new FileInputStream("xanadu.txt");
            out = new FileOutputStream("outagain.txt");
            int c;

            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

Java IO

Byte Stream




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

public class CopyBytes {
    public static void main(String[] args) throws IOException {

        FileInputStream in = null;
        FileOutputStream out = null;

        try {
            in = new FileInputStream("xanadu.txt");
            out = new FileOutputStream("outagain.txt");
            int c;

            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

Java IO

Character Stream

Class FileReader

java.lang.Object
    java.io.Reader
        java.io.InputStreamReader
            java.io.FileReader

Class FileWriter

java.lang.Object
    java.io.Writer
        java.io.OutputStreamWriter
            java.io.FileWriter

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

public class CopyCharacters {
    public static void main(String[] args) throws IOException {

        FileReader inputStream = null;
        FileWriter outputStream = null;

        try {
            inputStream = new FileReader("xanadu.txt");
            outputStream = new FileWriter("characteroutput.txt");

            int c;
            while ((c = inputStream.read()) != -1) {
                outputStream.write(c);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
    }
}
```

Java IO

Character Line Stream

Class BufferedReader

java.lang.Object
    java.io.Reader
        java.io.BufferedReader

Class PrintWriter

java.lang.Object
    java.io.Writer
        java.io.PrintWriter

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.IOException;

public class CopyLines {
    public static void main(String[] args) throws IOException {

        BufferedReader inputStream = null;
        PrintWriter outputStream = null;

        try {
            inputStream = new BufferedReader(new FileReader("xanadu.txt"));
            outputStream = new PrintWriter(new FileWriter("characteroutput.txt"));

            String l;
            while ((l = inputStream.readLine()) != null) {
                outputStream.println(l);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
    }
}
```

Java IO

Character Line Stream (Scanner)

```
import java.io.*;
import java.util.Scanner;

public class ScanXan {
    public static void main(String[] args) throws IOException {

        Scanner s = null;

        try {
            s = new Scanner(new BufferedReader(new FileReader("xanadu.txt")));

            while (s.hasNext()) {
                System.out.println(s.next());
            }
        } finally {
            if (s != null) {
                s.close();
            }
        }
    }
}
```

Java IO

Number Stream (Scanner)

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.Scanner;
import java.util.Locale;

public class ScanSum {
    public static void main(String[] args) throws IOException {

        Scanner s = null;
        double sum = 0;

        try {
            s = new Scanner(new BufferedReader(new FileReader("usnumbers.txt")));
            s.useLocale(Locale.US);

            while (s.hasNext()) {
                if (s.hasNextDouble()) {
                    sum += s.nextDouble();
                } else {
                    s.next();
                }
            }
        } finally {
            s.close();
        }

        System.out.println(sum);
    }
}
```

Java IO

Buffered Stream Read/Write

A API de Java IO permite melhorar a performance das operações de IO através de *“buferização”*

Leituras bufferizadas lêem dados de uma estrutura de memória chamada Buffer, e acionam API Nativa apenas quando o buffer estiver vazio.

Escritas bufferizadas escrevem dados para o Buffer, e quando este estiver vazio é que a API Nativa é acionada para escrever na saída.

```
inputStream = new BufferedReader(new FileReader("xanadu.txt"));
outputStream = new BufferedWriter(new FileWriter("characteroutput.txt"));
```

Java IO

Data Stream

DataStream permite leitura e escrita de dados primitivos (boolean, char, byte, short, int, long, float e double) e String

Todos os DataStream implementam a interface DataInput para entrada e DataOutput para a saída

Java IO

Data Stream

```
out = new DataOutputStream(new BufferedOutputStream(
                new FileOutputStream(dataFile)));
```

```
static final String dataFile = "invoicedata";

static final double[] prices = { 19.99, 9.99, 15.99, 3.99, 4.99 };
static final int[] units = { 12, 8, 13, 29, 50 };
static final String[] descs = {
    "Java T-shirt",
    "Java Mug",
    "Duke Juggling Dolls",
    "Java Pin",
    "Java Key Chain"
};
```

```
for (int i = 0; i < prices.length; i ++) {
    out.writeDouble(prices[i]);
    out.writeInt(units[i]);
    out.writeUTF(descs[i]);
}
```

Java IO

Data Stream

```
in = new DataInputStream(new
            BufferedInputStream(new FileInputStream(dataFile)));

double price;
int unit;
String desc;
double total = 0.0;
```

```
try {
    while (true) {
        price = in.readDouble();
        unit = in.readInt();
        desc = in.readUTF();
        System.out.format("You ordered %d" + " units of %s at $%.2f%n",
            unit, desc, price);
        total += unit * price;
    }
} catch (EOFException e) {
}
```

Java IO

Object Stream

Objetos podem ser *"serializados"*, permitindo desta forma que sejam gravados em arquivos ou enviados através de Socket (rede), e também recuperados a partir de arquivos e Sockets (rede)

Para tornar um objeto passível de serialização, a classe deve implementar a interface *Serializable*

Serializable é uma interface de *"marcação"*, ou seja, não existem métodos a serem implementados

```
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException
private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException;
private void readObjectNoData()
    throws ObjectStreamException;
```

Java IO

Object Stream

Para salvar objetos use a classe ObjectOutputStream

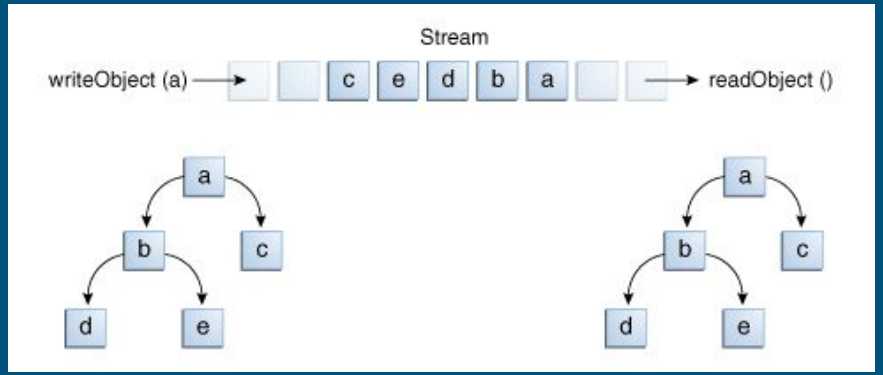
Para ler objetos use a classe ObjectInputStream

Class ObjectInputStream

java.lang.Object
    java.io.InputStream
        java.io.ObjectInputStream

Class ObjectOutputStream

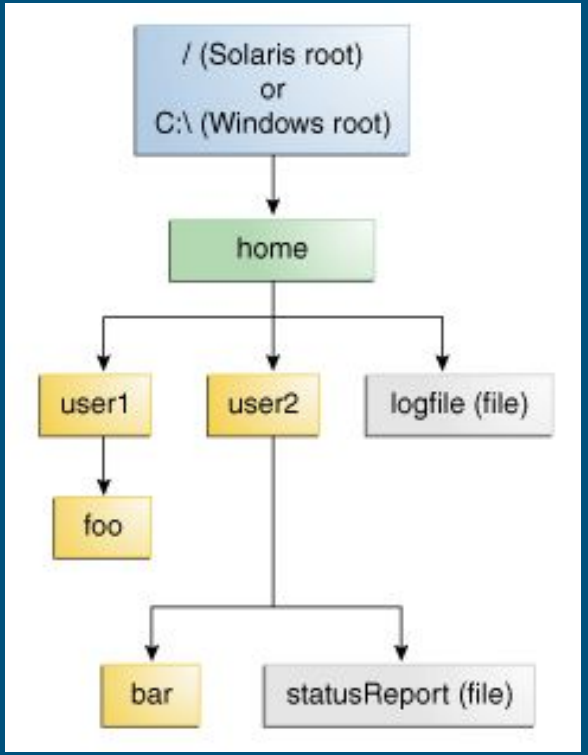
java.lang.Object
    java.io.OutputStream
        java.io.ObjectOutputStream



Java NIO

Path

Representa um caminho no Sistema de Arquivos



Java NIO

FileSystem Views

BasicFileAttributeView: Atributos básicos que deve ser suportado por todos os sistemas de arquivos

DosFileAttributeView: Atributos suportado pelo sistema de arquivo do DOS

PosixFileAttributeView: Estende o BasicFileAttributeView, acrescentando atributos suportado pelo padrão POSIX (Portable Operating System Interface for Unix)

FileOwnerAttributeView: Suportado por qualquer sistema de arquivo que implementa o conceito de *owner* de um arquivo.

AclFileAttributeView: Suporta leitura e atualização de arquivos ACL. O modelo NFSv4 ACL é suportado.

UserDefinedFileAttributeView: Suporta metadados em que o usuário é definido.

Java NIO

FileSystem Views

```
import java.nio.file.FileSystem;
import java.nio.file.FileSystems;
import java.util.Set;
…
FileSystem fs = FileSystems.getDefault();
Set<String> views = fs.supportedFileAttributeViews();

for (String view : views) {
      System.out.println(view);
}
```

Java NIO

FileSystem Views

```
FileSystem fs = FileSystems.getDefault();
for (FileStore store : fs.getFileStores()) {
   boolean supported = store.supportsFileAttributeView(BasicFileAttributeView.class);
   System.out.println(store.name() + " ---" + supported);
}
```

Java NIO

FileSystem Views

```
Path path = Paths.get("C:/rafaelnadal/tournaments/2009", "BNP.txt");

try {
    FileStore store = Files.getFileStore(path);
    boolean supported = store.supportsFileAttributeView("basic");
    System.out.println(store.name() + " ---" + supported);
} catch (IOException e) {
    System.err.println(e);
}
```

Java NIO

Symbolic e Hard Links

Atalhos para arquivos e diretórios

Hard Links podem ser criados apenas para arquivos e não para diretórios. Links Simbolicos podem ser criados para diretórios também

Hard Links não podem existir entre sistemas de arquivos, enquanto Links Simbolicos sim

O destino de Hard Link tem que existir, enquanto no Simbolico não é necessário

Remover um arquivo apontado pelo Hard Link não remove o Link e o mesmo continua retornando o conteúdo do arquivo. No caso do Simbolico, o link não é removido porém sem o arquivo original ele torna-se inútil

Java NIO

Symbolic e Hard Links

A remoção dos links não afeta o arquivo original

O Hard Link tem os mesmos atributos do arquivo original, enquanto o Soft Link não é tão restritivo

Java NIO

Symbolic e Hard Links

```
Path link = FileSystems.getDefault().getPath("rafael.nadal.1");
Path target= FileSystems.getDefault().getPath("C:/rafaelnadal/photos", "rafa_winner.jpg");

try {
    Files.createSymbolicLink(link, target);
    } catch (IOException | UnsupportedOperationException | SecurityException e) {
      if (e instanceof SecurityException) {
          System.err.println("Permission denied!");
      }
      if (e instanceof UnsupportedOperationException) {
          System.err.println("An unsupported operation was detected!");
      }
      if (e instanceof IOException) {
          System.err.println("An I/O error occurred!");
      }
System.err.println(e);
}
```

Java NIO

Symbolic e Hard Links

```
Path link = FileSystems.getDefault().getPath("rafael.nadal.4");
Path target = FileSystems.getDefault().getPath("C:/rafaelnadal/photos", "rafa_winner.jpg");

try {
    Files.createLink(link, target);
          System.out.println("The link was successfully created!");
    } catch (IOException | UnsupportedOperationException | SecurityException e) {
      if (e instanceof SecurityException) {
          System.err.println("Permission denied!");
      }
      if (e instanceof UnsupportedOperationException) {
          System.err.println("An unsupported operation was detected!");
      }
      if (e instanceof IOException) {
          System.err.println("An I/O error occured!");
      }
      System.err.println(e);
}
```

Java NIO

Files e Diretorios

Verificar existência de arquivos e diretórios

Verificar acessibilidade a um arquivo ou diretorio

Visibilidade de um arquivo

Criar, ler e escrever em arquivos

Criar arquivos temporários

Java NIO

Aspectos Avançados

Operações recursivas: FileVisitor

Watch Service API

Socket API

Asynchronous Channel API

...