

# Recapitulando: Ficheiros

UA.DETI.POO

# Operações de entrada/saída (I/O)

## Entrada

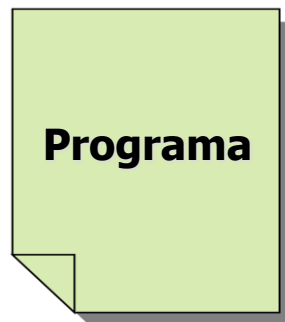
teclado



*leitura*



**Programa**



*escrita*



## Saída

monitor



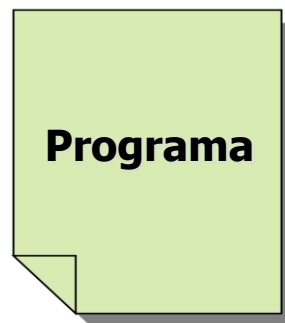
ficheiro



*leitura*



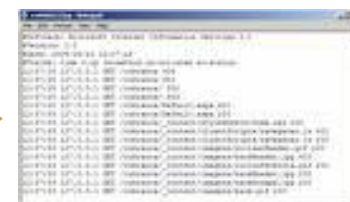
**Programa**



*escrita*



ficheiro



# java.io.File

---

- ❖ A classe *File* representa quer um nome de um ficheiro quer o conjunto de ficheiros num diretório
- ❖ Fornece informações e operações úteis sobre ficheiros e diretórios
  - `canRead`, `canWrite`, `exists`, `getName`, `isDirectory`, `isFile`, `listFiles`, `mkdir`, ...
- ❖ Exemplos:

```
File file1 = new File("io.txt");  
File file2 = new File("C:/tmp/", "io.txt");  
File file3 = new File("POO/Slides");  
  
if (!file1.exists()) { /* do something */ }  
if (!file3.isDirectory()) { /* do something */ }
```

# Ler dados ...

# ... usando java.util.Scanner

---

- ❖ Classe que facilita a leitura de tipos primitivos e de Strings a partir de uma fonte de entrada.

- Ler do teclado

```
Scanner sc1 = new Scanner(System.in);  
int i = sc1.nextInt();
```

- Ler de uma string

```
Scanner sc2 = new Scanner("really long\nString\n\t\tthat I want to pick apart\n");  
while (sc2.hasNextLine())  
    System.out.println(sc2.nextLine());
```

- Ler de um ficheiro

```
Scanner input = new Scanner(new File("words.txt"));  
while (input.hasNextLine())  
    System.out.println(input.nextLine());
```

# Escrita de dados

# Escrita de ficheiros de texto

---

## ❖ classe java.io.PrintWriter

- Permite-nos usar os métodos println e printf para escrever em ficheiros de texto.
- Formata os valores de tipos primitivos em texto, tal como quando impressos no écran.

```
public class FileWritingDemo {  
    public static void main(String[] args) throws IOException {  
        PrintWriter out = new PrintWriter(new File("file1.txt"));  
        out.println("Fim de semana na praia");  
        out.printf("Viagem: %d\nHotel: %d\n", 345, 1000);  
        out.close();  
    }  
}
```

# Aplicações de rede: Sockets

UA.DETI.POO



# O que são Aplicações de Rede?

---

- ❖ São programas que trocam informação através da rede.
  - Permite comunicar com outros computadores.
  - Mas também pode ser usado para comunicar dentro da propria máquina.
- ❖ As aplicações são (*tradicionalmente*) separadas em dois tipos:
  - Clientes
  - Servidores
- ❖ Para que a comunicação possa acontecer, é preciso saber o endereço do servidor, o protocolo de transporte (*TCP ou UDP*), o porto em que o servidor está a ouvir, e a lógica das mensagens trocadas (*protocolo da aplicação, pex., HTTP*).

# O que é um endereço?

---

- ❖ O endereço é um identificador único de um terminal na rede. Contudo, se o endereço for privado, este só tem de ser único dentro dessa rede (*i.e., só um terminal pode utilizar esse endereço nessa rede em todo o momento*), mas um terminal pode ter várias redes e vários endereços configurados.
  - Deve-se evitar colisões de endereços entre redes diferentes que têm de comunicar entre si, sob pena de ter erros de conectividade.
- ❖ Há três grandes tipos de endereço a considerar, apresentados pela ordem de popularidade: endereços por DNS (pex., localhost), endereços IPv4 (pex., 127.0.0.1), e endereços IPv6 (pex., ::1).
  - O endereço por DNS irá, na verdade, ser resolvido num endereço IPv4 ou IPv6 através do servidor de DNS configurado.

# O que é um protocolo de transporte?

---

- ❖ O protocolo de transporte serve para enviar dados entre dois endereços, um de source e outro de destino.
- ❖ Há dois grandes protocolos de transporte: TCP e UDP.
- ❖ TCP garante fiabilidade na ligação, retransmitindo pacotes perdidos, e ajustando a velocidade de envio à capacidade atual da ligação.
- ❖ UDP permite enviar informação de forma estritamente unidireccional, com menos overheads, mas também menos garantias de fiabilidade.

# O que é um porto?

---

- ❖ O porto serve para multiplexar o tráfego do mesmo protocolo de transporte entre dois endereços source e destino.
- ❖ Permite ter vários serviços desse protocolo expostos no mesmo servidor, são selecionados pelo número de **porto de destino**.
- ❖ Permite que um cliente tenha várias ligações ativas com o mesmo serviço, as ligações são identificadas pelo número de **porto de source**.

# O que é o protocolo da aplicação?

---

- ❖ O protocolo de aplicação é a construção das mensagens usadas pelo nosso programa para enviar informação/comandos entre um cliente e um servidor.
- ❖ O protocolo mais comum é o HTTP, que é usado para navegar na web e explorar várias APIs REST.
- ❖ Quando mandamos dados num no Java estamos a escrever diretamente no protocolo de aplicação, sendo o protocolo de transporte automaticamente gerado de acordo com a inicialização do nosso Socket.

# O que é um Socket?

---

- ❖ Socket é o canal de comunicação estabelecido entre duas máquinas, usando os seus endereços, protocolo de transporte, e porto.
- ❖ Pensem como um tubo do Super Mário, depois de estabelecido a informação entra numa ponta e, pouco depois, sai magicamente na outra.
- ❖ Por omissão os Sockets em Java são TCP, para UDP têm de usar a classe DatagramSocket.

# Como se faz isto em Java

---

- ❖ O Java tem o package `java.net.*` com classes que permitem abstrair o I/O através de Sockets de uma forma muito similar a lidar com ficheiros.
- ❖ Para além de TCP (*i.e.*, `Socket`) e UDP (*i.e.*, `DatagramSocket`) também permite estabelecer ligações diretamente em TCP + HTTP (*i.e.*, `HttpURLConnection`)
- ❖ Para simplicidade, vamos explorar apenas TCP.

# Código do Servidor:

(<https://www.baeldung.com/a-guide-to-java-sockets>)

```
public class GreetServer {
    private ServerSocket serverSocket;
    private Socket clientSocket;
    private PrintWriter out;
    private BufferedReader in;

    public void start(int port) {
        serverSocket = new ServerSocket(port);
        clientSocket = serverSocket.accept();
        out = new PrintWriter(clientSocket.getOutputStream(), true);
        in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        String greeting = in.readLine();
        if ("hello server".equals(greeting)) {
            out.println("hello client");
        }
        else {
            out.println("unrecognised greeting");
        }
    }

    public void stop() {
        in.close();
        out.close();
        clientSocket.close();
        serverSocket.close();
    }

    public static void main(String[] args) {
        GreetServer server=new GreetServer();
        server.start(6666);
    }
}
```



# Código do Cliente:

(<https://www.baeldung.com/a-guide-to-java-sockets>)

```
public class GreetClient {
    private Socket clientSocket;
    private PrintWriter out;
    private BufferedReader in;

    public void startConnection(String ip, int port) {
        clientSocket = new Socket(ip, port);
        out = new PrintWriter(clientSocket.getOutputStream(), true);
        in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
    }

    public String sendMessage(String msg) {
        out.println(msg);
        String resp = in.readLine();
        return resp;
    }

    public void stopConnection() {
        in.close();
        out.close();
        clientSocket.close();
    }
}
```

# Limitações

---

- ❖ Para que o servidor possa atender múltiplos pedidos em simultâneo seria necessário algum tipo de programação concorrente.
- ❖ Estratégias comuns incluem criar threads, utilizar operações assíncronas em métodos não bloqueantes, e programação por eventos.
- ❖ Todos estes tópicos serão abordados em muito mais detalhe nas disciplinas respetivas, o essencial a reter é ser possível e fácil fazer um programa em Java que comunica remotamente com outra máquina usando as abstrações de I/O já conhecidas.

# Sumário

---

- ❖ Fundamentais de comunicação em rede (básicos)
- ❖ java.net.\*
- ❖ Socket, DatagramSocket, HTTPURLConnection
  
- ❖ Ler informação da rede com um **BufferedReader**
- ❖ Escrever informação para a rede com **PrintWriter**
  
- ❖ Muitas outras classes e detalhes disponíveis
  - <https://docs.oracle.com/javase/8/docs/api/java/net/package-summary.html>
  - *(Disciplinas futuras)*