

# O que é um Sistema Distribuído?

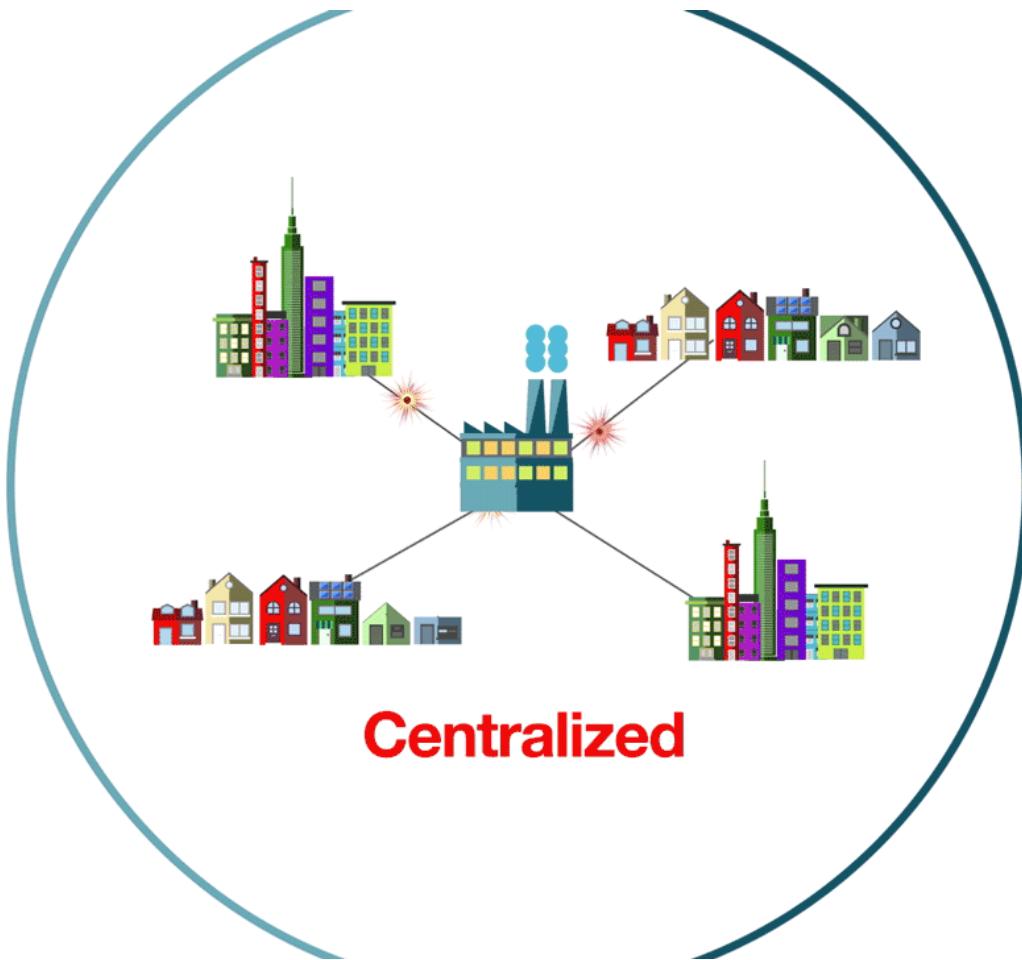
Nelson Rosa – nsr@cin.ufpe.br

# Objetivos da Aula

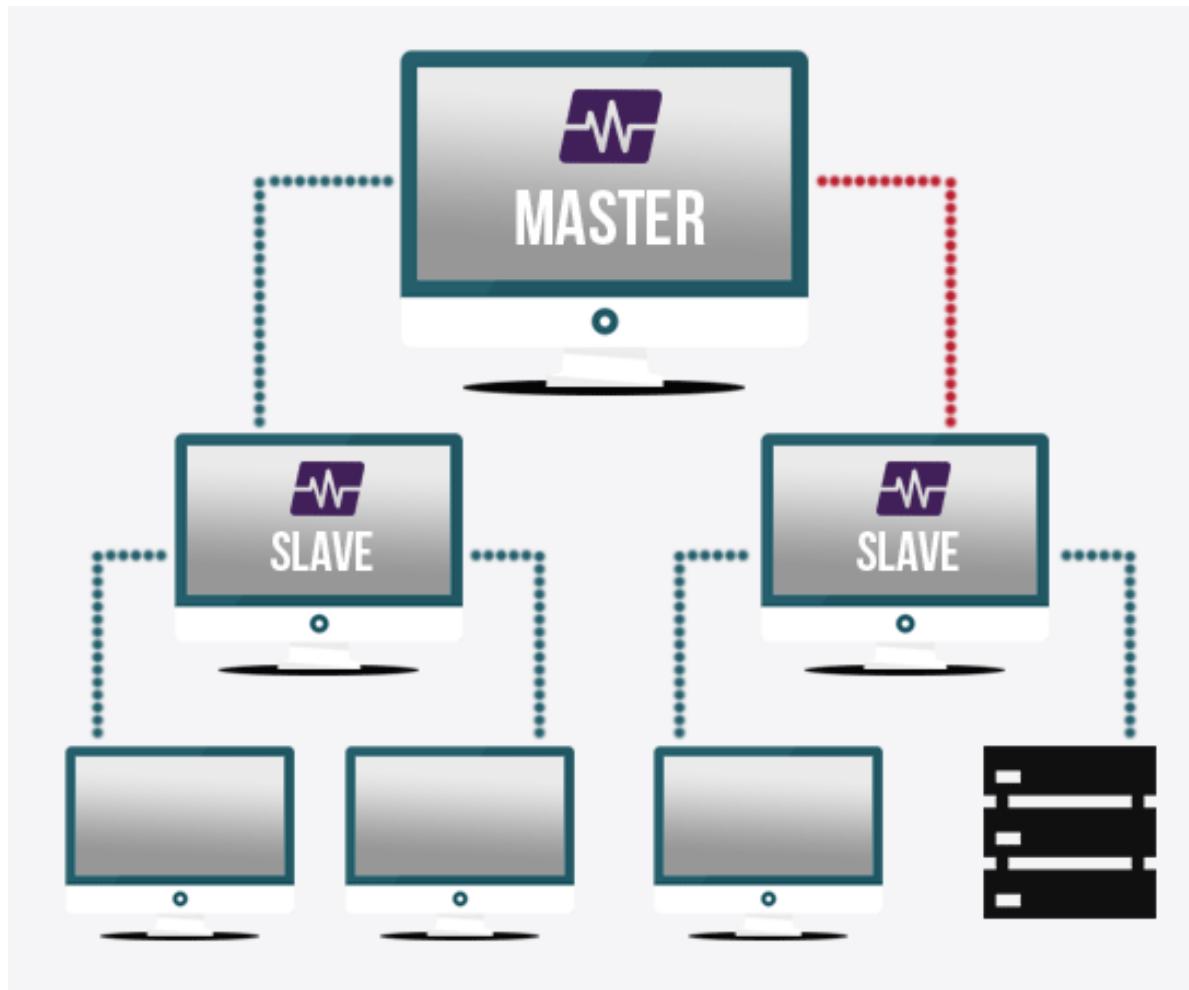
---

- **Apresentar definições de Sistemas Distribuídos**
  - O que é intuitivamente um SD?
  - O que nos dizem as definições clássicas de SD?
  
- **Realizar um passo-a-passo de como implementar um sistema distribuído**
  - Socket

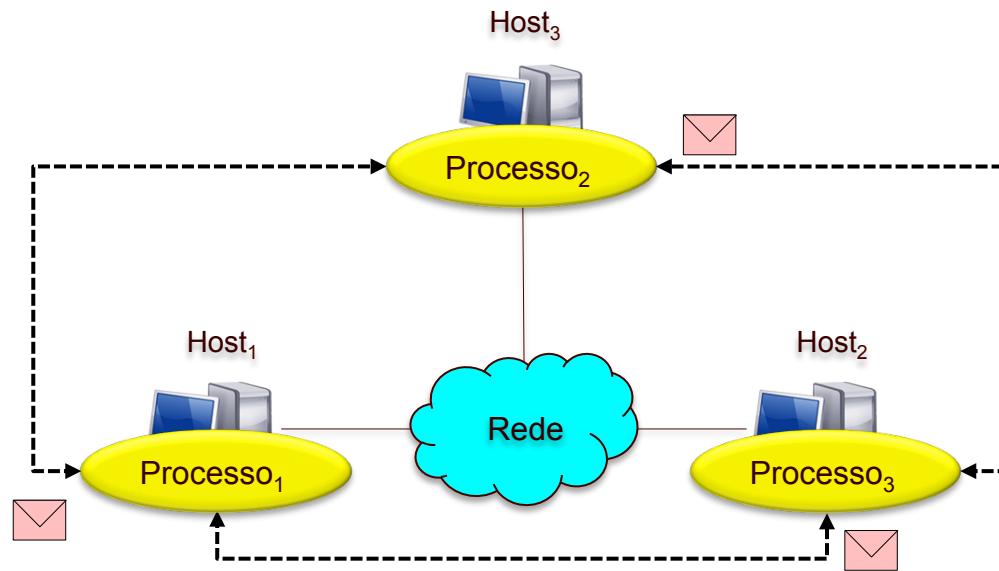
# Intuição sobre o que é um SD?



# Intuição sobre o que é um SD?



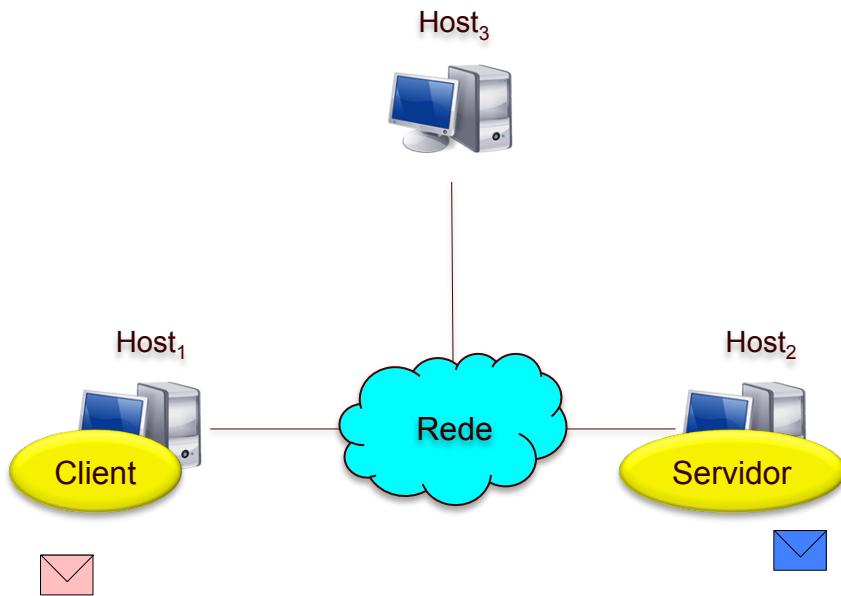
# Intuição sobre o que é um SD?



Mensagem (e.g., "sum (1,2)" ou "oi")

# Intuição sobre o que é um SD?

---



Mensagem, e.g., add (1,2)

Mensagem, e.g., 3

# Definições Clássicas e/ou Populares

---

- **Def. 1:** “Um sistema distribuído é um **sistema de informação** que possui um conjunto de **computadores independentes** que **cooperam** uns com os outros através de uma **rede de comunicação** para alcançar um objetivo específico”. (Arno Puder)
- **Def. 2:** “Um sistema distribuído é uma **coleção de computadores independentes** que aparecem aos seus usuários como um **sistema único e coerente**”. (Andrew Tanenbaum)
- **Def. 3:** “Um sistema distribuído consiste de uma **coleção de computadores autônomos** ligados por uma **rede** e equipado com um **software distribuído**” (George Coulouris)
- **Def. 4:** “Um sistema distribuído é aquele em que uma falha de um computador que você sequer sabia que existia faz com que o seu não funcione” (Leslie Lamport)

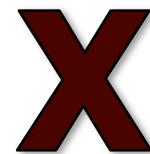
# Como construir um SD na prática?

---

**Inter Process  
Communication  
(e.g., socket)**



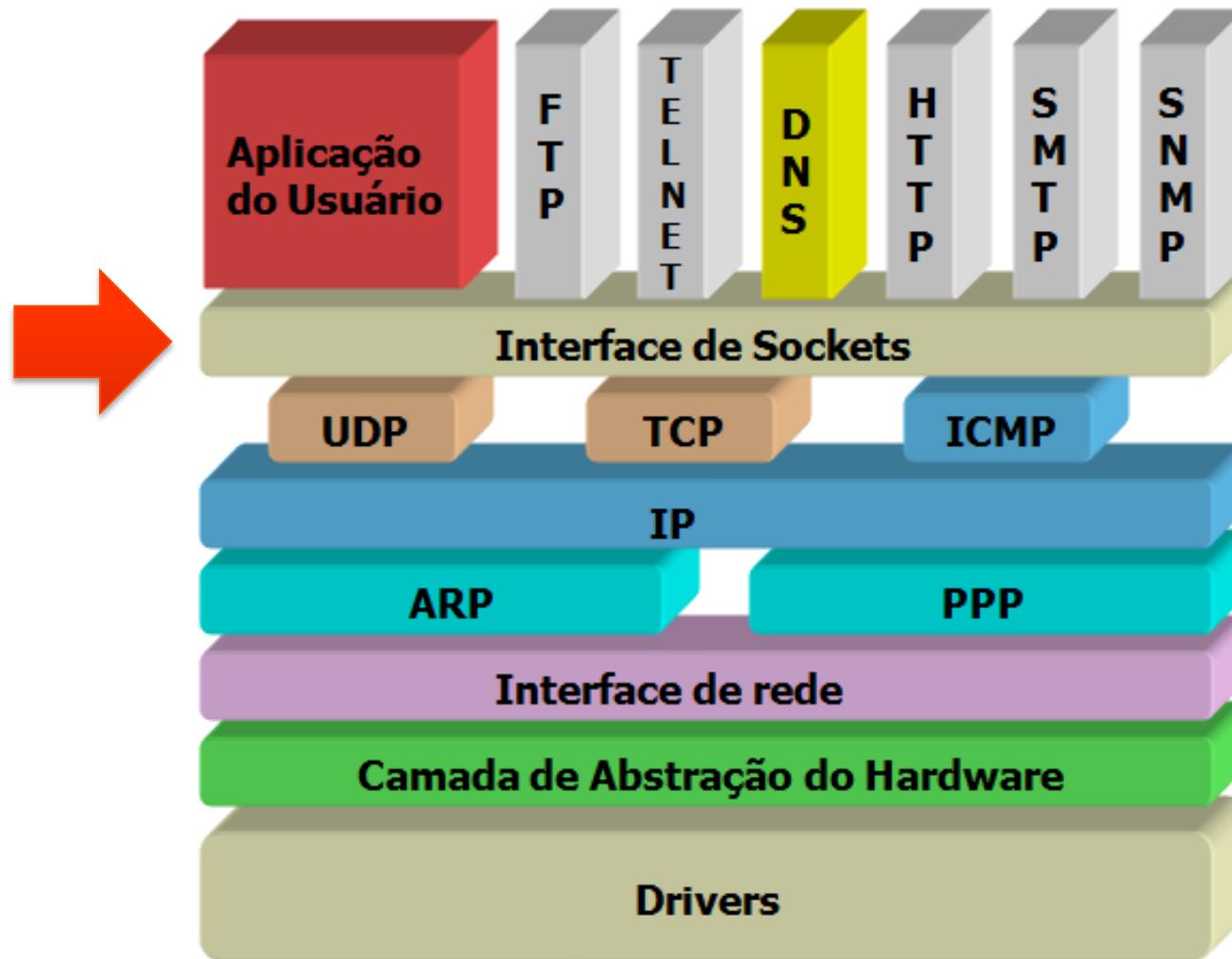
**Como  
Distribuir?**



**Middleware**

# SD na Prática:: Socket

---



## SD na Prática:: Socket

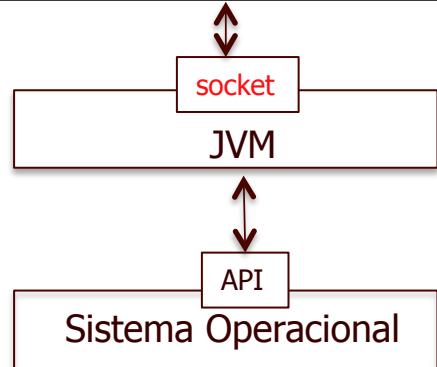
---



# SD na Prática:: Usando socket TCP

## Host 1

```
public class CalculatorClient { }
```



## Method Summary

All Methods	Static Methods	Instance Methods	Concrete Methods
Modifier and Type	Method and Description		
void	<code>bind(SocketAddress bindpoint)</code> Binds the socket to a local address.		
void	<code>close()</code> Closes this socket.		
void	<code>connect(SocketAddress endpoint)</code> Connects this socket to the server.		
void	<code>connect(SocketAddress endpoint, int timeout)</code> Connects this socket to the server with a specified timeout value.		
SocketChannel	<code>getChannel()</code> Returns the unique <code>SocketChannel</code> object associated with this socket, if any.		
InetAddress	<code>getInetAddress()</code> Returns the address to which the socket is connected.		
InputStream	<code>getInputStream()</code> Returns an input stream for this socket.		
boolean	<code>getKeepAlive()</code> Tests if <code>SO_KEEPALIVE</code> is enabled.		
InetAddress	<code>getLocalAddress()</code> Gets the local address to which the socket is bound.		

- ✓ A classe **Socket** tem 42 métodos.
- ✓ Variam de SO para SO.

# SD na Prática:: Usando socket TCP

**Passo 1: Abrir um socket**

**Passo 2: Associar um input/output stream para o socket**

**Passo 3: Escrever/Ler no stream**

**Passo 4: Fechar o stream**

**Passo 5: Fechar o socket**

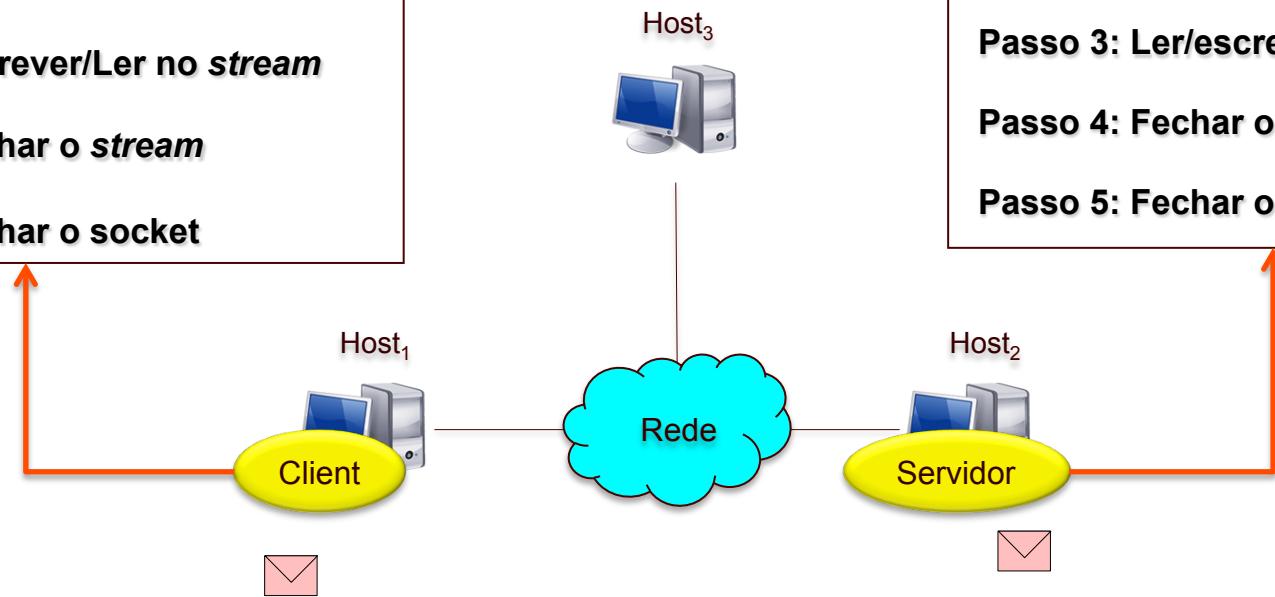
**Passo 1: Abrir um socket**

**Passo 2: Associar um input/output stream para o socket**

**Passo 3: Ler/escrever no stream**

**Passo 4: Fechar o stream**

**Passo 5: Fechar o socket**



# SD na Prática:: Usando socket TCP

---

## ■ No Servidor

- Precisa definir em que **porta** o servidor irá receber as mensagens
- O número da porta deve estar no intervalo **[0,65535]**
- Portas entre 0 e 1023 são **reservadas**
- Há 65536 portas TCP e 65536 portas UDP

## ■ No Cliente

- Precisa saber **onde** (ip) e em que **porta** (porta) o servidor está esperando receber as mensagens
  - ip = “151.161.1.1” ou “localhost”
  - porta = 1313

## ■ Toda comunicação envolve duas portas

- **Porta do Servidor:** você escolhe (e.g., 1313)
- **Porta do Cliente:** SO escolhe

# Socket:: TCP:: Passo 1:: Servidor

Passo 1: Abrir um socket

Passo 2: Associar um input/output *stream* para o socket

Passo 3: Escrever/Ler no *stream*

Passo 4: Fechar o *stream*

Passo 5: Fechar o socket

```
8 public class Server {  
9  
10    public static void main(String[] args) throws IOException {  
11  
12        // Step 1: open socket  
13        ServerSocket welcomeSocket = new ServerSocket(1313);  
14        Socket connectionSocket = welcomeSocket.accept();  
15    }
```

- ✓ **endpoint:** ip + porta
- ✓ **Socket:** É um *endpoint* para comunicação bidirecional entre duas máquinas.
- ✓ **ServerSocket:** *socket* usado para aguardar e aceitar conexões.
- ✓ **accept():** aguarda por uma conexão e cria um novo socket (*connectionSocket*)

# Socket:: TCP:: Passo 1:: Cliente

Passo 1: Abrir um socket

Passo 2: Associar um input/output *stream* para o socket

Passo 3: Escrever/Ler no *stream*

Passo 4: Fechar o *stream*

Passo 5: Fechar o socket



```
7 public class Client {  
8  
9     public static void main(String[] args) throws IOException, Throwable {  
10  
11         // Step 1: open socket  
12         Socket clientSocket = new Socket("localhost", 1313);  
13 }
```

- ✓ **localhost**: nome do *host* onde está o servidor.
- ✓ **1313**: número da porta onde o servidor está esperando pelas requisições.

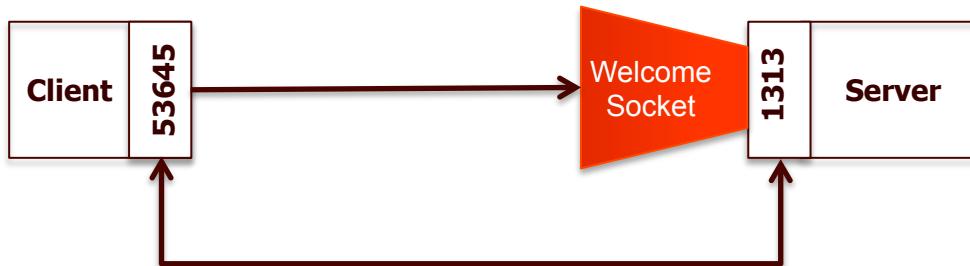
# Socket:: TCP:: Passo 1

Host<sub>1</sub>

```
7 public class Client {  
8     public static void main(String[] args) throws IOException, Throwable {  
9         // Step 1: open socket  
10        Socket clientSocket = new Socket("localhost", 1313);
```

Host<sub>2</sub>

```
8 public class Server {  
9     public static void main(String[] args) throws IOException {  
10        // Step 1: open socket  
11        ServerSocket welcomeSocket = new ServerSocket(1313);  
12        Socket connectionSocket = welcomeSocket.accept();  
13    }
```



# Socket:: TCP:: Passo 2

---

## ■ No Servidor

- Criar um **input stream** para receber dados do cliente
- Caso necessário, abrir um *output stream* para enviar dados para o cliente

## ■ No Cliente

- Criar um **output stream** para enviar dados para o servidor
- Caso necessário, abrir um *input stream* para receber dados do servidor

## ■ Há vários tipos de **streams** em Java

- ObjectInputStream, ObjectOutputStream
- DataInputStream, DataOutputStream

# Socket:: TCP:: Passo 2:: Servidor

Passo 1: Abrir um socket

Passo 2: Associar um input/output *stream* para o socket

Passo 3: Escrever/Ler no *stream*

Passo 4: Fechar o *stream*

Passo 5: Fechar o socket

```
9 public class Server {  
10  
11     public static void main(String[] args) throws IOException,  
12             ClassNotFoundException {  
13  
14         // Step 1: open socket  
15         ServerSocket welcomeSocket = new ServerSocket(1313);  
16         Socket connectionSocket = welcomeSocket.accept();  
17  
18         // Step 2: wrap the socket with a stream  
19         ObjectInputStream inFromClient = new ObjectInputStream(  
20             connectionSocket.getInputStream());  
21         ObjectOutputStream outToClient = new ObjectOutputStream(  
22             connectionSocket.getOutputStream());  
23 }
```

- ✓ **ObjectInputStream**: recebimento (leitura) de objetos e tipos de dados primitivos.
- ✓ **ObjectOutputStream**: envio (escrita) de objetos e tipos de dados primitivos.

# Socket:: TCP:: Passo 2:: Cliente

**Passo 1: Abrir um socket**

**Passo 2: Associar um input/output *stream* para o socket**

**Passo 3: Escrever/Ler no *stream***

**Passo 4: Fechar o *stream***

**Passo 5: Fechar o socket**

```
8 public class Client {  
9  
10    public static void main(String[] args) throws IOException, Throwable {  
11  
12        // Step 1: open socket  
13        Socket clientSocket = new Socket("localhost", 1313);  
14  
15        // Step 2: wrap the socket with a stream  
16        ObjectOutputStream outToServer = new ObjectOutputStream(clientSocket.getOutputStream());  
17        ObjectInputStream inFromServer = new ObjectInputStream(clientSocket.getInputStream());  
18
```

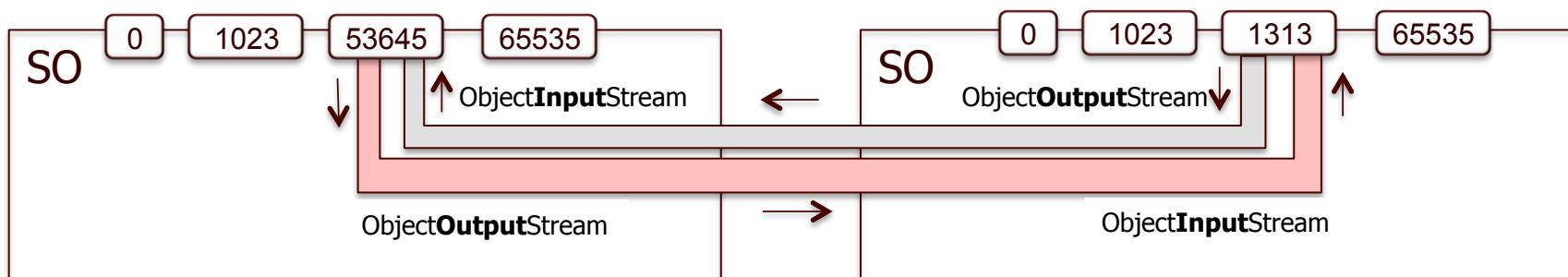
# Socket:: TCP:: Passo 2

Host<sub>1</sub>

```
8 public class Client {  
9  
10    public static void main(String[] args) throws IOException, Throwable {  
11        // Step 1: open socket  
12        Socket clientSocket = new Socket("localhost", 1313);  
13  
14        // Step 2: wrap the socket with a stream  
15        ObjectOutputStream outToServer = new ObjectOutputStream(clientSocket.getOutputStream());  
16        ObjectInputStream inFromServer = new ObjectInputStream(clientSocket.getInputStream());  
17  
18    }
```

Host<sub>2</sub>

```
9 public class Server {  
10  
11    public static void main(String[] args) throws IOException,  
12        ClassNotFoundException {  
13  
14        // Step 1: open socket  
15        ServerSocket welcomeSocket = new ServerSocket(1313);  
16        Socket connectionSocket = welcomeSocket.accept();  
17  
18        // Step 2: wrap the socket with a stream  
19        ObjectInputStream inFromClient = new ObjectInputStream(  
20            connectionSocket.getInputStream());  
21        ObjectOutputStream outToClient = new ObjectOutputStream(  
22            connectionSocket.getOutputStream());  
23    }
```



# Socket:: TCP:: Passo 3

---

## ■ **Servidor**

- Aguardar e receber as informações enviadas pelo cliente
- Processar as informações
- Caso necessário, enviar o resultado do processamento ao cliente
- Geralmente colocado em um *loop* infinito

## ■ **Cliente**

- Enviar informações para o servidor
- Caso exista, aguardar e receber o resultado do processamento do servidor

# Socket:: TCP:: Passo 3:: Servidor

Passo 1: Abrir um socket

Passo 2: Associar um input/output stream para o socket

Passo 3: Escrever/Ler no stream

Passo 4: Fechar o stream

Passo 5: Fechar o socket

```
9 public class Server {  
10  
11     public static void main(String[] args) throws IOException,  
12             ClassNotFoundException {  
13  
14         // Step 1: open socket  
15         ServerSocket welcomeSocket = new ServerSocket(1313);  
16         Socket connectionSocket = welcomeSocket.accept();  
17  
18         // Step 2: wrap the socket with a stream  
19         ObjectInputStream inFromClient = new ObjectInputStream(  
20             connectionSocket.getInputStream());  
21         ObjectOutputStream outToClient = new ObjectOutputStream(  
22             connectionSocket.getOutputStream());  
23  
24         // Step 3: write/read in the stream  
25         String msgFromClient = null;  
26         msgFromClient = (String) inFromClient.readObject();  
27  
28         String msgToClient = null;  
29         msgToClient = msgFromClient.toUpperCase();  
30         outToClient.writeObject(msgToClient);
```

# Socket:: TCP:: Passo 3:: Cliente

```
8 public class Client {  
9  
10    public static void main(String[] args) throws IOException, Throwable {  
11  
12        // Step 1: open socket  
13        Socket clientSocket = new Socket("localhost", 1313);  
14  
15        // Step 2: wrap the socket with a stream  
16        ObjectOutputStream outToServer = new ObjectOutputStream(clientSocket.getOutputStream());  
17        ObjectInputStream inFromServer = new ObjectInputStream(clientSocket.getInputStream());  
18  
19        // Step 3: write/read in the stream  
20        String msgToServer = "frase a ser modificada";   
21        outToServer.writeObject(msgToServer);  
22  
23        String msgFromServer = null;   
24        msgFromServer = (String) inFromServer.readObject();  
25    }
```

# Socket:: TCP:: Passos 4 & 5:: Servidor e Cliente

Passo 1: Abrir um socket

Passo 2: Associar um input/output *stream* para o socket

Passo 3: Escrever/Ler no *stream*

Passo 4: Fechar o *stream*

Passo 5: Fechar o socket

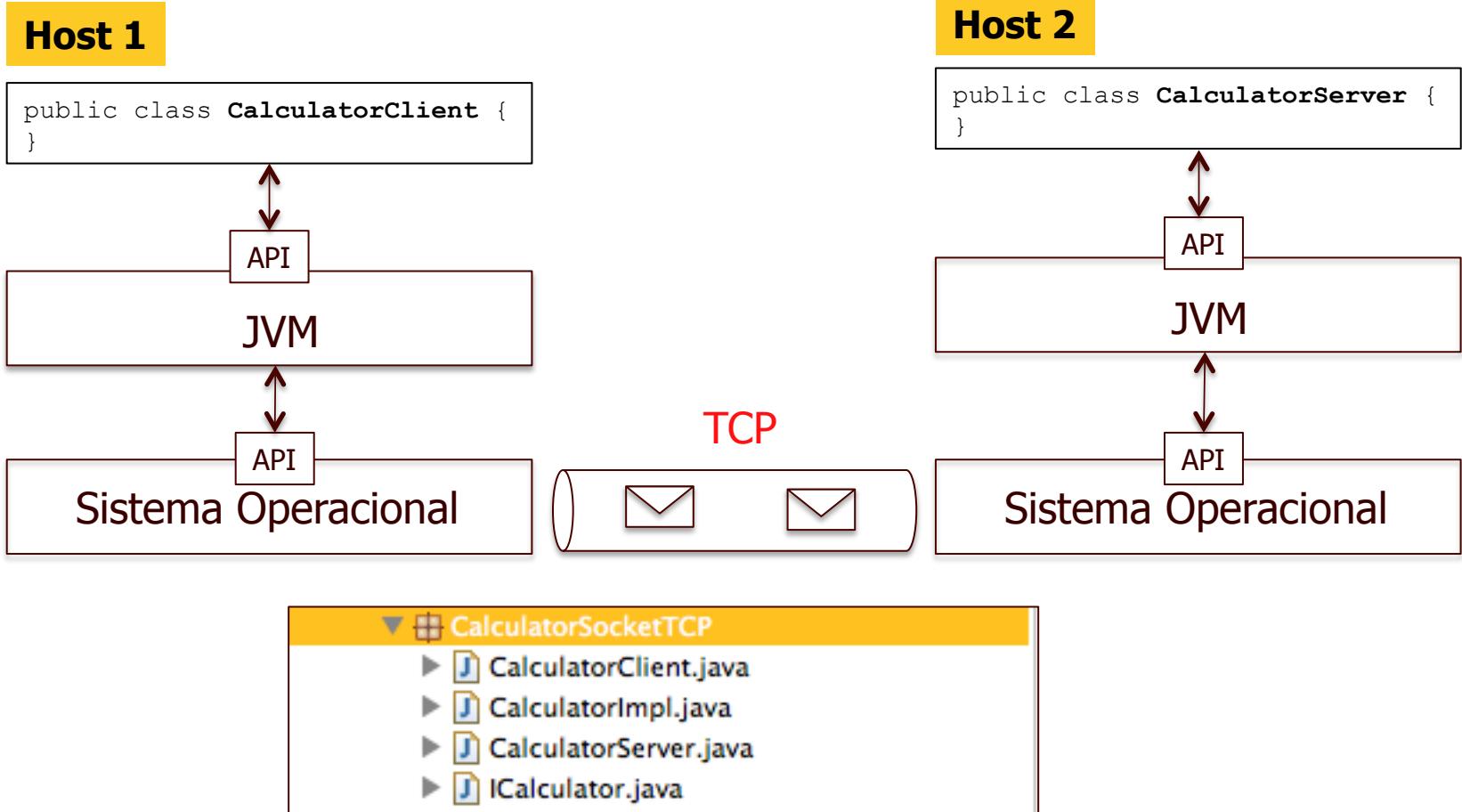
```
32      // Step 4: close the streams
33      inFromClient.close();
34      outToClient.close();
35
36      // Step 5: close the socket
37      connectionSocket.close();
38      welcomeSocket.close();
```

Servidor

```
28      // Step 4: close the streams
29      outToServer.close();
30      inFromServer.close();
31
32      // Step 5: close the socket
33      clientSocket.close();
```

Cliente

# Exemplo:: Calculadora:: Distribuída:: Socket:: TCP



# Exemplo:: Calculadora:: Distribuída:: Socket:: Interface

```
3 public interface ICalculator {  
4  
5     float sum (float p1, float p2);  
6     float sub (float p1, float p2);  
7     float div (float p1, float p2);  
8     float mul (float p1, float p2);  
9 }
```

## Interface

## Implementação

```
1 package CalculatorSocket;  
2  
3 public class CalculatorImpl implements ICalculator {  
4  
5     @Override  
6     public float sum(float p1, float p2) {  
7         return p1+p2;  
8     }  
9  
10    @Override  
11    public float sub(float p1, float p2) {  
12        return p1-p2;  
13    }  
14  
15    @Override  
16    public float div(float p1, float p2) {  
17        return p1/p2;  
18    }  
19  
20    @Override  
21    public float mul(float p1, float p2) {  
22        return p1 * p2;  
23    }  
24  
25 }
```

# Exemplo:: Calculadora:: Distribuída:: Socket:: TCP:: Servidor

```
9 public class CalculatorServer {  
10  
11     public static void main(String[] args) throws IOException,  
12         ClassNotFoundException {  
13  
14         // create sockets + streams  
15         ServerSocket welcomeSocket = new ServerSocket(1313);  
16         Socket connectionSocket = welcomeSocket.accept();  
17         ObjectOutputStream outToClient = new ObjectOutputStream(  
18             connectionSocket.getOutputStream());  
19         ObjectInputStream inFromClient = new ObjectInputStream(  
20             connectionSocket.getInputStream());  
21  
22         // receive request from client  
23         String rcvMsg = null;  
24         rcvMsg = (String) inFromClient.readObject();  
25  
26         // process request  
27         CalculatorImpl calculator = new CalculatorImpl();  
28         float r = 0;  
29         String opName = rcvMsg.substring(0, 3);  
30         String[] pars = rcvMsg.substring(rcvMsg.indexOf("(") + 1,  
31             rcvMsg.indexOf(")").split(",");  
32         float p1 = Float.parseFloat(pars[0]);  
33         float p2 = Float.parseFloat(pars[1]);  
34  
35         if (opName.equals("sub"))  
36             r = calculator.sub(p1, p2);  
37  
38         // send response to client  
39         String repMsg = Float.toString(r);  
40         outToClient.writeObject(repMsg);  
41         outToClient.flush();  
42  
43         // close sockets + streams  
44         connectionSocket.close();  
45         welcomeSocket.close();  
46         outToClient.close();
```

- ✓ Suporte à uma única operação ("sub")
- ✓ Porta 1313 (TCP)
- ✓ Limitações?
- ✓ E se quisermos usar UDP?

# Exemplo:: Calculadora:: Distribuída:: Socket:: TCP:: Cliente

```
8 public class CalculatorClient {  
9  
10    public static void main(String[] args) throws IOException,  
11        | ClassNotFoundException {  
12  
13        // CalculatorImpl calculator = new CalculatorImpl();  
14        // System.out.println(calculator.sub(1,3));  
15  
16        // create socket + streams  
17        Socket clientSocket = new Socket("localhost", 1313);  
18        ObjectOutputStream outToServer = new ObjectOutputStream(  
19            clientSocket.getOutputStream());  
20        ObjectInputStream inFromServer = new ObjectInputStream(  
21            clientSocket.getInputStream());  
22  
23        // send request to server  
24        outToServer.writeObject("sub(1,3)");  
25        outToServer.flush();  
26  
27        // receive response from server  
28        String repMsg = null;  
29        repMsg = (String) inFromServer.readObject();  
30  
31        System.out.println(">>> " + repMsg);  
32  
33        // close sockets + streams  
34        clientSocket.close();  
35        outToServer.close();  
36        inFromServer.close();  
37  
38    }  
39 }  
40 }
```

## Exemplo:: Calculadora:: Linhas de Código

---

	Interface (*)	Cliente (*)	Servidor (*)
Centralizada	6	6	14
Distribuída (TCP)	6	20	57
Distribuída (UDP)	6	19	55

# Códigos

---

## ■ Disponíveis em

- [https://www.dropbox.com/sh/7u9f4zd9uip8xzj/  
AABOFpGWOpCLt\\_Skic\\_Wlsjka?dl=0](https://www.dropbox.com/sh/7u9f4zd9uip8xzj/AABOFpGWOpCLt_Skic_Wlsjka?dl=0)

---

# **Fim dos Slides**