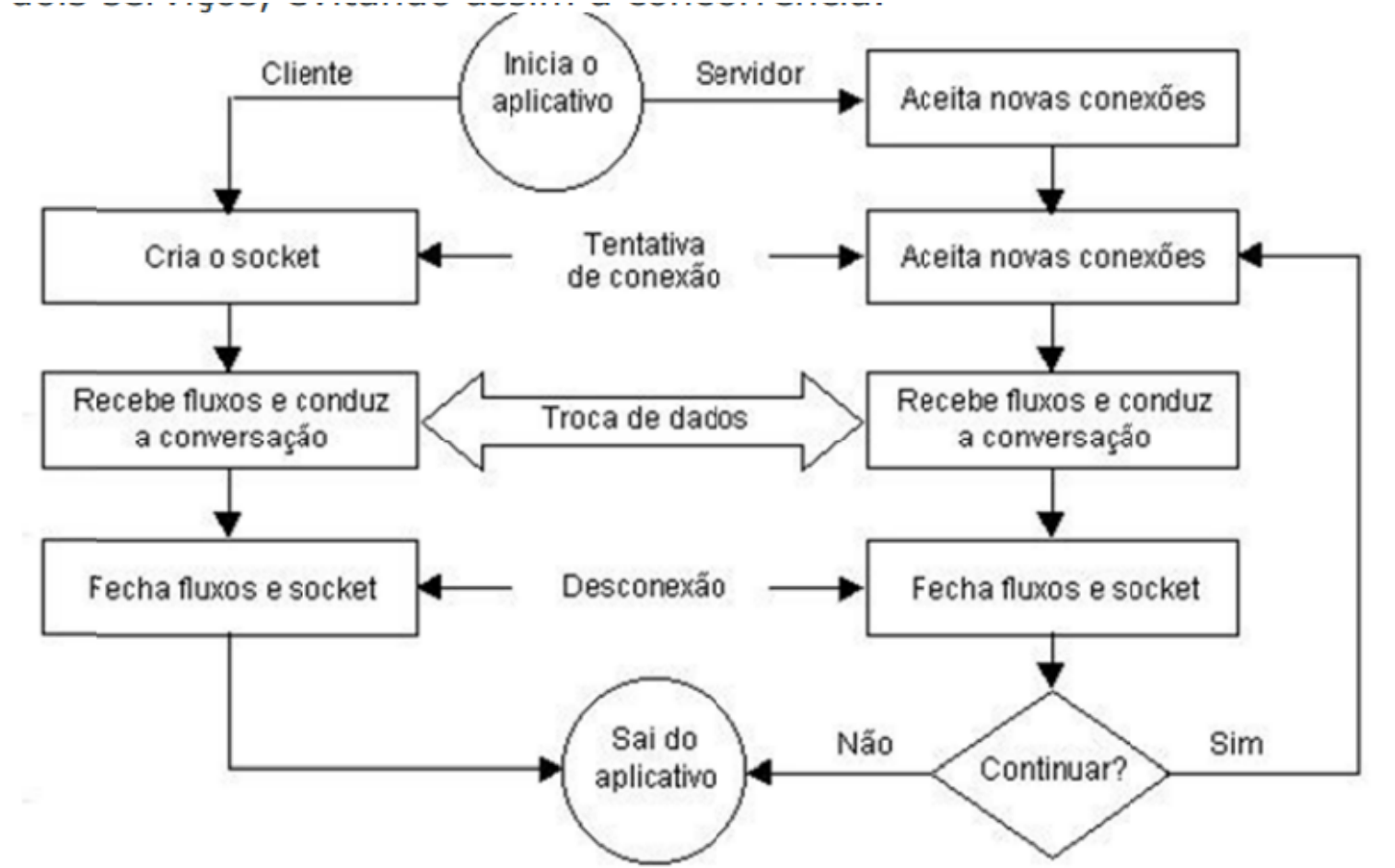
Sockets com Java

- Uma aplicação que utiliza sockets normalmente é composta por uma parte servidora e diversos clientes.
- O cliente solicita determinado serviço ao servidor, o servidor processa e devolve a informação ao cliente.
- Muitos serviços podem ser disponibilizados numa mesma máquina, sendo então diferenciados não só pelo endereço IP, mas também pelo número da porta.

Fluxo de troca de dados com socket



Criando um servidor

- Primeiramente deve-se importar o pacote java.net
- Em seguida, instanciar um objeto do tipo ServerSocket
  - Responsável por atender pedidos via rede e em uma determinada porta.
- Após estabelecer a conexão, um objeto do tipo Socket deve ser criado para manter a comunicação entre o cliente e o servidor.
- Ex.: ServerSocket server = new ServerSocket(12345);
  - cria o ServerSocket, que irá esperar conexões na porta 12345

Criando um servidor

- Em seguida criamos um objeto Socket, o qual irá tratar da comunicação com o cliente, assim que um pedido de conexão chegar ao servidor e a conexão for aceita:
- Socket client = server.accept();

Endereços IP

- Cada máquina conectada a uma rede possui um endereço IP único de maneira que possa ser identificada na rede. A classe InetAdress nos permite obter informações sobre um computador conectado a rede. Os principais métodos desta classe são os seguintes:
  - **getAddress():** retorna um array de bytes contendo o endereço IP
  - **getHostAddress():** retorna uma Stringcontendo o endereço IP no formato 999.999.999.999
  - **getHostName():** Dado um array de bytes contendo o endereço IP de um host, este método retorna uma String com o nome do host

O Protocolo TCP

- Quando necessitamos troca confiável de informações, isto é, quando é necessária a confirmação de recebimento da mensagem enviada, devemos utilizar o protocolo TCP(*Transmission Control Protocol*).
- A programação do TCP com sockets utiliza *streams*, o que simplifica muito o processe de leitura e envio de dados pela rede.
- Streams são objetos Java que permitem obter dados de qualquer fonte de entrada, seja o teclado, um arquivo ou até mesmo um fluxo de bytes recebidos pela rede (o que é o nosso caso).

Um primeiro servidor TCP

- Vamos montar um servidor TCP que permite a seus clientes solicitarem a data e a hora atuais do servidor.
- Nesse primeiro exemplo apenas um cliente pode ser atendido por vez.

Código do servidor TCP

```
public class ServidorTCPBasico {
  public static void main(String[] args) {
    try {
      // Instancia o ServerSocket ouvindo a porta 12345
      ServerSocket servidor = new ServerSocket(12345);
      System.out.println("Servidor ouvindo a porta 12345");
      while(true) {
        // o método accept() bloqueia a execução até que
        // o servidor receba um pedido de conexão
        Socket cliente = servidor.accept();
        System.out.println("Cliente conectado: " +
cliente.getInetAddress().getHostAddress());
```

1º

2º

3º

Cliente solicita conexão

Servidor exibe msg com endereço IP do cliente conectado

Código do servidor TCP

```
        ObjectOutputStream saida = new
ObjectOutputStream(cliente.getOutputStream());
        saida.flush();
        saida.writeObject(new Date());
        saida.close();
        cliente.close();
      }
    }
    catch(Exception e) {
       System.out.println("Erro: " + e.getMessage());
    }
    finally {...}
  }
}
```

Servidor aceita a conexão, envia um objeto Date ao cliente e encerra a conexão

Código do cliente TCP

```
public class ClienteTCPBasico {
  public static void main(String[] args) {
    try {
      Socket cliente = new Socket("paulo",12345);
      ObjectInputStream entrada = new
ObjectInputStream(cliente.getInputStream());
      Date data_atual = (Date)entrada.readObject();
    JOptionPane.showMessageDialog(null,"Data recebida do servidor:"
+ data_atual.toString());
    entrada.close();
    System.out.println("Conexão encerrada");
  }
  catch(Exception e) {
    System.out.println("Erro: " + e.getMessage());
  }
 }
}
```

Cliente solicita conexão com o servidor

O cliente recebe o objeto do servidor e faz o cast necessário, em seguida exibe na tela as informações de data;

Mensagens do servidor

Data e hora recebidas pelo cliente

Protocolo UDP

- Quando a troca de mensagens pode ser não-confiável podemos usar o protocolo UDP (*User Datagram Protocol*).
- É de responsabilidade da aplicação receptora a remontagem dos pacotes na ordem correta e a solicitação de reenvio de pacotes que não foram recebidos.

Protocolo UDP

- Exemplo – conversa entre dois usuários:
  - Erika: Ola Paulo
  - Paulo: Ola Erika
  - Erika: Como você está?
  - Paulo: Tudo bem e vc? (este pacote foi perdido)
    - No exemplo acima o TCP enviaria novamente o pacote e o usuário da outra ponta deveria ficar esperando. Não faria nenhum sentido isso tratando-se de uma ligação telefônica. Este é um típico caso onde o UDP se aplica perfeitamente.

Um primeiro servidor UDP

- Como vimos, o UDP envia os pacotes sem esperar por uma resposta do receptor. Este protocolo pode ser útil em situações como o envio de pacotes multimídia, por exemplo, ou um serviço de voz sobre ip, o que é muito comum.

Código do servidor UDP

- Nosso servidor UDP envia mensagens para os clientes de uma determinada rede local. Perceba neste exemplo que no UDP o cliente também aguarda mensagens que poderão ser enviadas pelo servidor, ou seja, mantém um DatagramSocket em uma determinada porta. Por exemplo, o seguinte trecho cria o DatagramSocket que irá esperar mensagens na porta 12346.

```
DatagramSocket serverdgram = new DatagramSocket(12346);
```

Classe Remetente UDP

```
public class RemetenteUDP {
  public static void main(String[] args) {
    if(args.length != 3) {
        System.out.println("Uso correto: <Nome da maquina> <Porta>
<Mensagem>");
       System.exit(0);
    }
    try {
      //Primeiro argumento é o nome do host destino
      InetAddress addr = InetAddress.getByName(args[0]);
      int port = Integer.parseInt(args[1]);
      byte[] msg = args[2].getBytes();
      //Monta o pacote a ser enviado
       DatagramPacket pkg = new DatagramPacket(msg,msg.length, addr,
port);
```

Classe Remetente UDP

```
        // Cria o DatagramSocket que será responsável por enviar a
mensagem
        DatagramSocket ds = new DatagramSocket();
        //Envia a mensagem
        ds.send(pkg);
        System.out.println("Mensagem enviada para: " +
addr.getHostAddress() + "\n" +
         "Porta: " + port + "\n" + "Mensagem: " + args[2]);
        //Fecha o DatagramSocket
        ds.close();
    }
    catch(IOException ioe) {...}
  }
}
```

Classe Receptor UDP

```
public class ReceptorUDP {
    public static void main(String[] args) {
        if(args.length != 1) {
            System.out.println("Informe a porta a ser ouvida");
            System.exit(0);
        }
        try {
            //Converte o argumento recebido para inteiro (numero da porta)
            int port = Integer.parseInt(args[0]);
            //Cria o DatagramSocket para aguardar mensagens, neste momento o
método fica bloqueando
            //até o recebimento de uma mensagem
            DatagramSocket ds = new DatagramSocket(port);
```

Classe Receptor UDP

```
            //Preparando o buffer de recebimento da mensagem
            byte[] msg = new byte[256];
            //Prepara o pacote de dados
            DatagramPacket pkg = new DatagramPacket(msg, msg.length);
            //Recebimento da mensagem
            ds.receive(pkg);

JOptionPane.showMessageDialog(null,newString(pkg.getData()).trim(), "Mensagem
recebida",1);
            ds.close();
        }
        catch(IOException ioe) {...}
    }
}
```

Remetente UDP em ação (enviando mensagem ao receptor)

Receptor sendo inicializado para receber mensagens na porta 1234

Mensagem UDP recebida pelo receptor