INSTITUTO FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA

Introdução à Programação Android

Prof. Samir Bonho

Aula 10

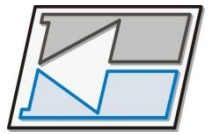
Florianópolis, 30 de Abril de 2014

Sumário

- ✓ Sockets
- ✓ Fontes:
  - ✓ Android Developers. Disponível em http://developer.android.com/

INSTITUTO FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA

Socket TCP em Android



Usando Sockets em Java

- Pacote `java.net`;
- Principais classes:
  - TCP: `Socket` e `ServerSocket`;
  - UDP: `DatagramPacket` e `DatagramSocket`;
  - *Multicast*: `DatagramPacket` e `MulticastSocket`.
- Este pacote também contém classes que fornecem suporte a manipulação de URLs e acesso a serviços HTTP, entre outras coisas.

Sockets TCP

- Clientes e servidores são **diferentes**.
  - Servidor usa a classe `ServerSocket` para "escutar" uma porta de rede da máquina a espera de requisições de conexão.

```
ServerSocket sSocket = new ServerSocket(12345,5);
```

  - Clientes utilizam a classe `Socket` para requisitar conexão a um servidor específico e então transmitir dados.

```
Socket cSocket = new Socket("213.13.45.2",12345);
```

Servidor TCP

- Inicialmente, um servidor deve criar um *socket* que associe o processo a uma porta do *host*;

```
ServerSocket sSocket = new ServerSocket(porta,backlog);
```

porta: número da porta que o *socket* deve esperar requisições;

backlog: tamanho máximo da fila de pedidos de conexão que o sistema pode manter para este *socket*.

Servidor TCP

- O *backlog* permite que requisições sejam enfileiradas (em estado de espera) enquanto o servidor está ocupado executando outras tarefas.
- As requisições são processadas uma a uma
- Se uma requisição de conexão chegar ao *socket* quando esta fila estiver cheia, a conexão é negada.

Servidor TCP

- O método **accept** é usado para retirar as requisições da fila.
  - Fica bloqueado até que um cliente solicite uma requisição

```
Socket socket = sSocket.accept();
```

- O método **accept** retorna uma conexão com o cliente

Servidor TCP

- Quando do recebimento da conexão, o servidor pode interagir com o cliente através da leitura e escrita de dados no *socket*.
- A comunicação em si é feita com o auxílio de classes tipo *streams*, que são classes do pacote `java.io`.
  - *Stream* de Leitura:

```
DataInputStream in = new
  DataIntputStream(cliente.getInputStream());
```

    Este *stream* tem vários métodos *read* (ver pacote `java.io`).
  - *Stream* de Escrita:

```
DataOutputStream out = new
DataOutputStream(socket.getOutputStream());
```

    Este *stream* tem vários métodos *write* (ver pacote `java.io`).

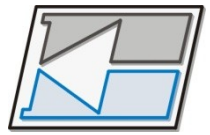
Servidor TCP

- Encerramento da conexão.
  - Com um cliente em específico:

```
socket.close();
```

  - Do *socket* servidor (terminando a associação com a porta do servidor):

```
sSocket.close();
```



Exemplo: servidor Java (TCP)

```
class ServerThread implements Runnable {

public void run() {
Socket socket = null;
try {
serverSocket = new ServerSocket(SERVERPORT);
} catch (IOException e) {
e.printStackTrace();
}
while (!Thread.currentThread().isInterrupted()) {

        try {

        socket = serverSocket.accept();
…
```

Cria socket de aceitação na porta especificada

Espera, no socket de aceitação, por contato do cliente

Exemplo: servidor Java (TCP)

```
public CommunicationThread(Socket clientSocket) {

this.clientSocket = clientSocket;

try {

this.input = new BufferedReader(new
InputStreamReader(this.clientSocket.getInputStrea
m()));

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

Cria stream de saída ligado ao socket

Cliente TCP

- **Inicialmente**, o cliente deve criar um *socket* especificando o endereço e a porta do serviço a ser acessado:

```
Socket socket = new Socket(host, porta);
```

host é endereço ou nome do servidor

porta é o número da porta em que o servidor está escutando

Esta chamada representa a requisição de uma conexão com o servidor. Se o construtor for executado sem problemas, a conexão está estabelecida.

- **A partir daí**, da mesma forma que no servidor, o cliente pode obter os *streams* de entrada e saída.

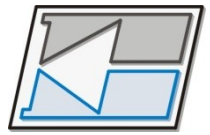
```
DataInputStream in = new
  DataIntputStream(cliente.getInputStream());
DataOutputStream out = new
  DataOutputStream(socket.getOutputStream());
```

- **Ao final,** o *socket* é fechado da mesma forma que no servidor.

Exemplo: cliente Java (TCP)

```
class ClientThread implements Runnable {

@Override
public void run() {

try {
        InetAddress serverAddr =
InetAddress.getByName(SERVER_IP);

socket = new Socket(serverAddr, SERVERPORT);

} catch (UnknownHostException e1) {
        e1.printStackTrace();
} catch (IOException e1) {
        e1.printStackTrace();
}
…
```

Cria stream de entrada ligado ao socket



Exemplo: cliente Java (TCP)

Cria stream de entrada

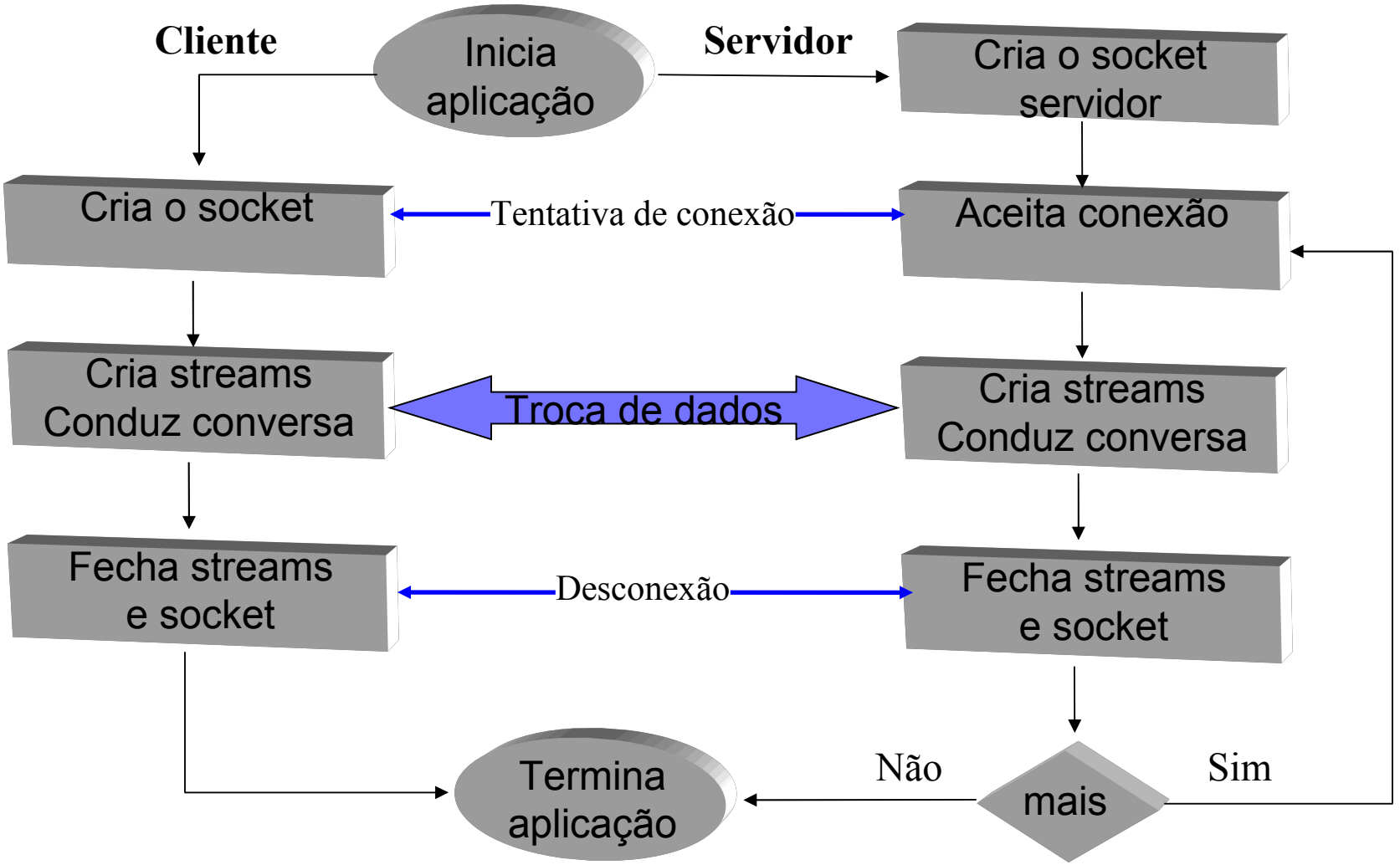
Envia linha para o servidor

```
public void onClick(View view) {
try {
EditText et = (EditText) findViewById(R.id.EditText01);
String str = et.getText().toString();
PrintWriter out = new PrintWriter(new BufferedWriter(
new OutputStreamWriter(socket.getOutputStream())),
true);
            out.println(str);
} catch (UnknownHostException e) {
            e.printStackTrace();
} catch (IOException e) {
            e.printStackTrace();
} catch (Exception e) {
            e.printStackTrace();
}
```

Resumo – Sockets TCP

- Servidor
  - Cria o *socket* servidor e aguarda conexão
  - Usa método accept() para pegar novas conexões
  - Cria streams entrada/saída para o *socket* da conexão
  - Faz a utilização dos *streams* conforme o protocolo
  - Fecha os *streams*
  - Fecha *socket* da conexão
  - Repete várias vezes
  - Fecha o *socket* servidor
- Cliente
  - Cria o socket com conexão cliente
  - Associa *streams* de leitura e escrita com o socket
  - Utiliza os *streams* conforme o protocolo do servidor
  - Fecha os *streams*
  - Fecha o *socket*

Resumo – Sockets TCP



Exercícios