

[Anterior](#) • [Trilha I](#) • [Próximo](#) • [comentar](#)

Busca:

Lição 2 - Sockets de servidor e de cliente

- **Objetivo(s):** Ensinar como criar sockets TCP.
- **Direitos autorais e licença:** Veja notas de direitos autorais e licença no final da lição.

Conteúdo

- [2.1 - Definição de Socket](#)
- [2.2 - Sockets usando TCP](#)
- [2.3 - "Hello World" com Sockets TCP](#)
- [2.4 - Direitos autorais e licença](#)
- [2.5 - Comentários](#)

2.1 - Definição de Socket

Podemos afirmar que todas as aplicações de rede usam (mesmo que de maneira transparente) a programação de *sockets*.

Um *socket* é uma ligação entre o processo de aplicação e um protocolo de transporte fim-a-fim (UDP ou TCP). É explicitamente criado, usado e liberado por APIs e segue o paradigma cliente/servidor discutido anteriormente.

Cada *socket*, ao ser criado deve ser vinculado a uma **porta**, que nada mais é que um ponto de interação entre uma aplicação e o sistema operacional da máquina, sendo representada por um endereço numérico. Após ser criado, o *socket* pode ser manipulado com as demais primitivas da lista abaixo:

- *Socket*: Cria um novo ponto de comunicação.
- *Bind*: Amarra um endereço a um *socket*.
- *Listen*: Anuncia disposição em atender conexões.
- *Accept*: Bloqueia até uma requisição de conexão chegar.
- *Connect*: Tenta estabelecer uma conexão.
- *Send*: Envia dados usando a conexão.
- *Receive*: Recebe dados usando da conexão.
- *Close*: Fecha a conexão.

💡 É importante conhecer a função de todas as primitivas, mas grande parte delas é chamada de maneira automática e transparente na API de Java, ou seja, o programador precisa dominar apenas as primitivas principais.

2.2 - Sockets usando TCP

Na programação com *Sockets* usando TCP, o processo servidor deve antes estar em execução aguardando contato do cliente. Quando contatado pelo cliente, o TCP do servidor cria um *socket* novo para que o processo servidor possa se comunicar com o cliente, o que permite que o servidor converse com múltiplos clientes. Endereço IP e porta origem são usados para distinguir os clientes.

Para criar o *socket* servidor, deve ser usada a classe *ServerSocket* do pacote *java.net*. O código abaixo cria um *socket* servidor que ficará esperando conexões na porta 9876. Note que o que determina a porta é o número passado no construtor ao se criar o objeto.

```
ServerSocket socketRecepcao = new ServerSocket(9876);
```

💡 O código acima pode lançar exceções do tipo *java.io.IOException* e *java.lang.SecurityException*. Apesar de não ser indicado, podemos colocar o código acima dentro de um único bloco *try/catch* usando no *catch* a classe *Exception*.

O *socket* criado no código acima servirá apenas para esperar conexões dos clientes, o que é feito através da chamada à primitiva *accept*, conforme o código abaixo:

```
socketRecepcao.accept();
```

Enquanto o *socket* servidor é criado usando a classe *ServerSocket*, para criação do cliente deve-se usar a classe *Socket*, também do pacote *java.net*. Isso é feito de maneira diferente do lado cliente e do lado servidor. No lado cliente o objeto é criado explicitamente, conforme o código abaixo:

```
Socket socketCliente = new Socket("localhost", 9876);
```

⚠️ Note que o cliente deve saber o nome da máquina (no nosso exemplo *localhost*) e a porta onde o *socket* servidor está executando.

No lado servidor o *socket*, representando o cliente, é criado de maneira transparente como resposta a uma chamada à primitiva *accept*.

```
Socket socketConexao = socketRecepcao.accept();
```

Após serem usados, os recursos devem ser liberados e os *sockets* fechados, através da primitiva *close*.

Do lado cliente:

```
socketCliente.close();
```

Do lado servidor:

```
socketConexao.close();
```

2.3 - "Hello World" com Sockets TCP

Nesta seção será dado um exemplo simples de uso de *sockets* para ilustrar os conceitos vistos na seção anterior. Por enquanto a única ação realizada é a conexão. A comunicação entre os *sockets* será explicada mais adiante no tutorial.

Do lado cliente temos o código abaixo:

```
import java.net.Socket;

public class ClienteTCP {

    public static void main(String argv[]) throws Exception {
        Socket socketCliente = new Socket("localhost", 9876);
        System.out.println("Socket cliente e criado e finalizando...");
        socketCliente.close();
    }
}
```

Baixe o código-fonte acima neste link: <http://wiki.marceloakira.com/pub/GrupoJava/SocketsDeServidorEDeCliente/ClienteTCP.java>

Do lado servidor é suficiente colocar o código abaixo. O método *getPort* usado com o *socket* de conexão serve para obter a porta usada para o *socket* criado: a porta 9876 é do *socket* servidor, haverá uma porta diferente para o *socket* criado para o cliente.

```
import java.net.ServerSocket;
import java.net.Socket;

public class ServidorTCP {
    public static void main(String argv[]) {
        try {
            ServerSocket socketRecepcao = new ServerSocket(9876);
            System.out.println("Servidor esperando conexão na porta 9876");
            Socket socketConexao = socketRecepcao.accept();
            System.out.println("Conexão estabelecida na porta " + socketConexao.getPort());
            socketConexao.close();
            socketRecepcao.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Baixe o código-fonte acima neste link: <http://wiki.marceloakira.com/pub/GrupoJava/SocketsDeServidorEDeCliente/ServidorTCP.java>

Contudo, o código acima apresenta um problema: o servidor atende somente um cliente e depois é finalizado. Ao conceituarmos servidor vimos que este deve estar ativo o tempo todo e atender vários clientes. Dessa forma, vamos melhorar o código acima para que isso seja feito. Podemos colocar o código responsável pela criação do *socket* cliente dentro de um laço, gerando o código abaixo:

```
import java.net.ServerSocket;
import java.net.Socket;

public class ServidorTCP {
    public static void main(String argv[]) {
        try {
            ServerSocket socketRecepcao = new ServerSocket(9876);
            System.out.println("Servidor esperando conexão na porta 9876");


            while (true) {
                Socket socketConexao = socketRecepcao.accept();
                System.out.println("Conexão estabelecida na porta " + socketConexao.getPort());
                socketConexao.close();
            }

            socketRecepcao.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Baixe o código-fonte acima neste link: <http://wiki.marceloakira.com/pub/GrupoJava/SocketsDeServidorEDeCliente/ServidorTCP.java>

Agora o servidor irá executar constantemente, pois a condição do laço sempre é verdadeira e, dessa forma, irá atender vários clientes.

Para testar a aplicação baixe o [Servidor](#) e o [Cliente](#) da aplicação. Execute primeiro o servidor e depois o cliente na mesma máquina.

 Você poderá executar o exemplo em máquinas diferentes, basta modificar o cliente trocando "localhost" pelo nome da máquina onde o servidor está executando.

2.4 - Direitos autorais e licença

- **Autor(es):** Raphael de Aquino Gomes
- **Direito Autoral:** Copyright © Sistemas Abertos
- **Licença:** Esta obra está licenciada sob uma [Licença Creative Commons](#).



[← Anterior](#) • [Trilha I](#) [↑](#) • [Próximo →](#)

2.5 - Comentários

Adicionar