Sockets

Sockets

- Os computadores ganham muito mais importância quando conectados entre si para trocar informações.
- A troca de dados entre computadores de uma mesma rede é realizada através de sockets.
- Um socket permite que um computador receba ou envie dados para outros computadores da mesma rede.
- A classe SOCKET define o funcionamento dos sockets em Java.

```
Socket socket = new Socket("184.72.247.119", 1000);
```

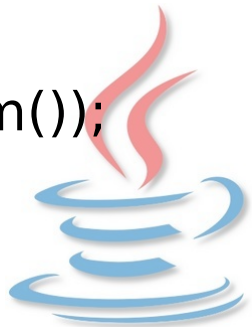
Sockets

- Um dos construtores da classe SOCKET recebe o **ip** e a **porta** da máquina que queremos nos conectar.
- Após a conexão através do socket ser estabelecida, podemos criar um objeto da classe PRINTSTREAM e outro da classe SCANNER associados ao socket para facilitar o envio e o recebimento dados respectivamente.

```
Socket socket = new Socket("184.72.247.119", 1000);

PrintStream saida = new PrintStream(socket.getOutputStream());

Scanner entrada = new Scanner(socket.getInputStream());
```



Server Sockets

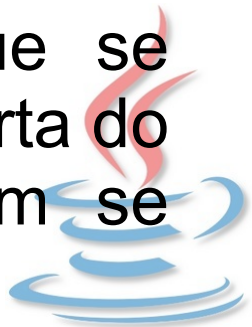
- Um server socket é um tipo especial de socket. Ele deve ser utilizado quando desejamos que uma aplicação seja capaz de aguardar que outras aplicações possivelmente em outras máquinas se conectem a ela.
- A classe ServerSocket define o funcionamento de um server sockets

```
ServerSocket severSocket = new ServerSocket(1000);

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

Server Sockets

- Um dos construtores da classe ServerSocket recebe a porta que será utilizada pelas aplicações que querem estabelecer uma conexão com a aplicação do server socket.
- O método ACCEPT() espera alguma aplicação se conectar na porta do server socket.
- Quando isso acontecer, o método ACCEPT() cria um novo socket em outra porta associado à aplicação que se conectou para realizar a troca de dados e liberar a porta do server socket para outras aplicações que desejem se conectar.



Server Sockets

- Se uma aplicação deseja permitir que diversas aplicação se conectem a ela então é necessário chamar várias vezes o método ACCEPT().
- Este método pode ser colocado em um laço.

```
ServerSocket severSocket = new ServerSocket(1000);

while(true) {

    Socket socket = serverSocket.accept();

}
```

- Cada iteração do laço acima estabelece uma conexão nova com uma aplicação cliente.

Exercício

1. Crie um projeto no eclipse chamado **Sockets**.

2. Crie o código de uma aplicação servidora.

```
public class Servidor {

   public static void main(String[] args) {
      try{
         System.out.println("[Criando Servidor...]");
         ServerSocket servidor = new ServerSocket(1234);
         System.out.println("[Servidor operando na porta
1234]");
         while(true){
            Socket cliente = servidor.accept();
            System.out.println("[Conexao aberta de :"+
cliente.getInetAddress().toString()+"]");
```

Exercício

```
System.out.println("[Enviando Dados...]");
            ObjectOutputStream saida = new
ObjectOutputStream(cliente.getOutputStream());
            saida.flush();//enviando cabeçalho de preparo do
outro endpoint
            saida.writeObject("Servidor Básico Conectado");
            saida.writeObject("Dados
conexão:"+cliente.toString());
            saida.writeObject("Tchau !");
            System.out.println("[Dados enviados]");
            saida.writeObject("EOT");
            cliente.close();
            System.out.println("[Conexao encerrada]");
         }
      }catch(Exception e){
         System.out.println("Erro !\n"+e.getMessage());
      }
   }
}
```

Exercício

3. Crie o código de uma aplicação cliente.

```
public class Cliente extends Thread{

   public static void main(String[] args) {
      try{
         String str = JOptionPane.showInputDialog("Informe
o número IP");
         Socket cliente = new Socket(str,1234);
         System.out.println("[Conexao aceita de:"+
cliente.getInetAddress().toString()+"]");
         System.out.println("[Recebendo Mensagens...]");
         ObjectInputStream entrada = new
ObjectInputStream(cliente.getInputStream());
         String msg;
```

Exercício

```
do{
                Thread.sleep(2000);
                msg=(String)entrada.readObject();
                System.out.println(msg);
            }while(!msg.equals("EOT"));
            cliente.close();
            System.out.println("[Conexao encerrada]");
        }catch(Exception e){
            System.out.println("Erro!\n"+e.getMessage());
        }
    }
}
```

Exercício

4. Agora vamos executar a classe Servidor para que ele fique aguardando as conexões.

5. Agora vamos executar a classe Cliente que verificará se existe um servidor para a conexão e depois irá executar os comando que foram elaborados para verificar a conexão.

6. Podemos analisar no console tudo que foi executado e tanto na parte do servidor quando na parte do cliente. Isso irá nos ajudar a compreender um pouco sobre como transitar informações pela rede.