

Laboratório III

Programação Java

romualdomrc@gmail.com



Ementa

- Implementação de algoritmos utilizando linguagem orientada a objetos.
- Estrutura de dados básica e avançada.
- Implementação dos principais conceitos de orientação a objetos: definição de classes e instanciação de objetos, encapsulamento, herança, polimorfismo, interfaces, tratamento de exceções, utilização de coleções.
- IDEs de desenvolvimento avançado de aplicativos desktop, manipulação de componentes de interface gráfica (propriedades e eventos relacionados).
- Conexão com bancos de dados relacionais.
- Geração de documentação.



Conteúdo

- Ambiente de Desenvolvimento.
- Desenvolvimento de aplicação desktop: componentes visuais, propriedades, métodos.
- Tratamento de exceções: definir exceções, descrever o uso, descrever categorias das exceções, identificar exceções comuns, escrever o código para gerenciar suas próprias exceções.



Conteúdo

- Fluxo de dados e arquivos (streams, manipulação de arquivos): usar a versão Streams (Fluxo) do pacote de Java, construir e usar Fluxo de I/O (I/O Streams), distinguir "Readers" e "Writers" , construir e usar Streams, compreender como criar suas próprias classes de processamento de Fluxo (Streams), ler, escrever, e atualizar dados em arquivos de acesso aleatório, usar a interface "Serialization" para codificar o estado de um objeto em um Fluxo de I/O e implementar a persistência do objeto.
- JavaDoc: exercitar a utilização da ferramenta JavaDoc para a geração de documentação de API.



Conteúdo

- Fundamentos de JDBC: entender o que são drivers, diferenciar ODBC, JDBC e DRIVERMANAGER, desenvolver uma aplicação Java para conexão com Banco de Dados, criar um objeto a partir da classe "Statement", utilizar os métodos executeUpdate e executeQuery da classe "Statement", saber configurar o ODBC da Microsoft para criar uma fonte de dados, saber utilizar os métodos das classes DataBaseMetaData e ResultSetMetaData, saber criar as "Prepared Statement", entender e aplicar o conceito de transação.



Referências Bibliográficas



Bibliografia

DEITEL, H.M., DEITEL, P.J. Java 2, Como Programar. 4^a Edição. Porto Alegre: Bookman, 2003



KATHY SIERRA BERT BATES, Use a Cabeça! Java, 1^a Edição. Alta Books, 2005

Outras Referências

Documentação *on-line* disponível em <http://java.sun.com/>

Programação Orientada a Objetos com Java. Uma introdução prática usando o BlueJ. David J. Barnes , Michael Kölling. Editora: Pearson Prentice Hall
Bluej: www.bluej.org



Introdução



O paradigma da Orientação a Objetos

- *Um paradigma é uma forma de abordar um problema.*
- O paradigma da orientação a objetos surgiu no fim dos anos 60.
- Hoje em dia, praticamente suplantou o paradigma anterior, o *paradigma estruturado...*



O paradigma da Orientação a Objetos

Alan Kay, um dos pais do paradigma da orientação a objetos, formulou a chamada **analogia biológica**.

“Como seria um sistema de software que funcionasse como um ser vivo?”



O paradigma da Orientação a Objetos

Cada “célula” interagiria com outras células através do envio de mensagens para realizar um objetivo comum.

Adicionalmente, cada célula se comportaria como uma unidade autônoma.

De uma forma mais geral, Kay pensou em como construir um sistema de software a partir de agentes autônomos que interagem entre si.

Com isso, ele estabeleceu os princípios da **orientação a objetos**.



Orientação a Objetos - Princípios

Tudo é um objeto.

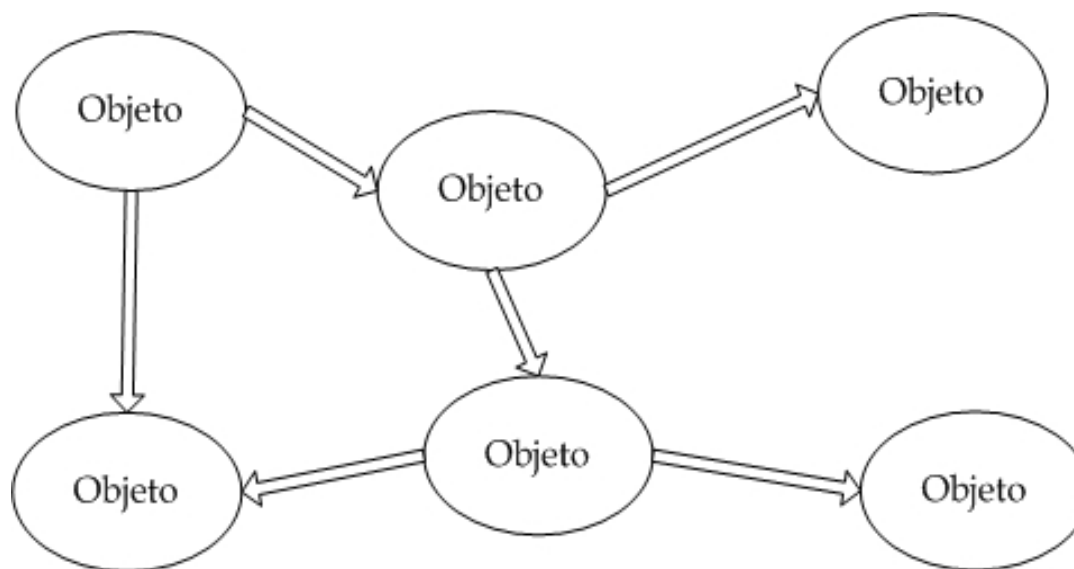
Pense em um objeto como uma super variável: ele armazena dados, mas você também pode fazer requisições a esse objeto, pedindo que ele faça operações sobre si próprio. Em teoria, você pode representar qualquer elemento conceitual no problema que você está tentando resolver (cachorros, livros, sócios, empréstimos, etc.) como um objeto no seu programa.



Orientação a Objetos - Princípios

- Um programa é uma coleção de objetos dizendo uns aos outros o que fazer.

Para fazer uma requisição a um objeto você “manda uma mensagem” para este objeto. Mais concretamente, você pode pensar em uma mensagem como sendo uma chamada de um procedimento ou função pertencente a um objeto em particular.



Conceitos e Princípios da OO

- Conceitos
 - Classe
 - Objeto
 - Mensagem
- Princípios
 - Encapsulamento
 - Polimorfismo
 - Generalização (Herança)
 - Composição



Características imperativas básicas de Java

Estrutura de um Programa
Tipos Primitivos
Estruturas de Controle



Estrutura mínima de um programa em Java

```
class <nome> {  
    public static void main (<parametro>)  
    {    <declarações>  
        <comandos>  
    }  
}
```

Onde main: método por onde se inicia a execução
public: qualificador de acesso
static: indica que main se aplica à classe
void: indica que main não retorna um valor



Exemplo

```
import java.util.Scanner;
class LeImprime {
/** Lê e imprime um string */
public static void main (String[] args) {
    Scanner le = new Scanner(System.in);
    String nome;
    nome = le.next();
    System.out.println (nome);
}
}
```



Tipos Primitivos

boolean	true ou false
char	caracteres (Unicode)
byte	inteiro (8 bits)
short	inteiro (16 bits)
int	inteiro (32 bits)
long	inteiro (64 bits)
float	ponto flutuante (32 bits)
double	ponto flutuante (64 bits)



Booleanos (boolean)

- Literais: true e false
- Operadores relacionais:
 > >= < <= == !=
- Operadores lógicos:
 && (and) || (or) ! (not)
- Precedência: unários, aritméticos, relacionais, lógicos (primeiro && e depois ||)



Strings (String)

- Não é um tipo primitivo e sim uma classe
- Literais: `""` `"a"` `"DI \n UFJF \n"` ...
- Operadores: `+` (concatenação)

ex.: `"março" + " de " + 98 = "março de 98"`

Note a conversão de inteiro para string

Há uma conversão implícita para todos os tipos primitivos



Mais operadores sobre strings

- Comparação (igualdade) de dois strings *a* e *b*
a.equals(b) ou *b.equals(a)*
- Tamanho de um string *a*
a.length()



Comandos básicos: atribuição

Forma geral:

- `<identificador> = <expressão>`

ex.: `x = 12.7; i = i + 1; st = "Recife"`

- Formas concisas de in(de)cremento:

`i += 1 i++` equivale a `i = i + 1`

`i -= 1 i--` equivale a `i = i - 1`



Condicional: if-else

Forma geral:

```
if (<expressão-booleana>
    <bloco-de-comandos1>
[else
    <bloco-de-comandos2>]
```

Onde: <bloco-de-comandos> é uma sequência de comandos entre { e }.



Exemplo: maior entre dois numeros

```
class Maior {  
    public static void main (String[] args) {  
        int x, y;  
        x = Util.readInt ();  
        y = Util.readInt ();  
        if (x > y )  
            System.out.println (x);  
        else  
            if (y > x)  
                System.out.println (y);  
            else  
                System.out.println ("x = y");  
    }  
}
```



Repetição: while

Forma geral:

```
while (<expressão-booleana>)  
    <bloco-de-comandos>
```



Exemplo: soma de 1 a n

...

```
int i, n, s;  
n = Util.readInt ();  
i = 1; s = 0;  
while (i <= n) {  
    s = s + i;  
    i = i + 1;  
}  
System.out.println(s);
```

...



Alguns conceitos básicos de orientação a objetos e Java

*Objeto,
Atributo,
Método,
Classe e
Encapsulamento*



Classes, objetos e mensagens

- O mundo real é formado de coisas.
- Na terminologia de orientação a objetos, estas coisas do mundo real são denominadas *objetos*.
- Seres humanos costumam agrupar os objetos para entendê-los.
- A descrição de um grupo de objetos é denominada **classe de objetos**, ou simplesmente de **classe**.



O que é uma classe?

- Uma classe é um *molde* para objetos. Diz-se que um objeto é uma *instância* de uma classe.
- Uma classe é uma **abstração** das características **relevantes** de um grupo de coisas do mundo real.
 - Na maioria das vezes, um grupo de objetos do mundo real é muito complexo para que todas as suas características e comportamento sejam representados em uma classe.



Programação Orientada a Objetos

Foco nos **dados** (objetos) do sistema, não nas **funções**

Estruturação do programa é baseada nos **dados**, não nas **funções**

As **funções** mudam mais do que os **dados**



Objeto Conta Bancária

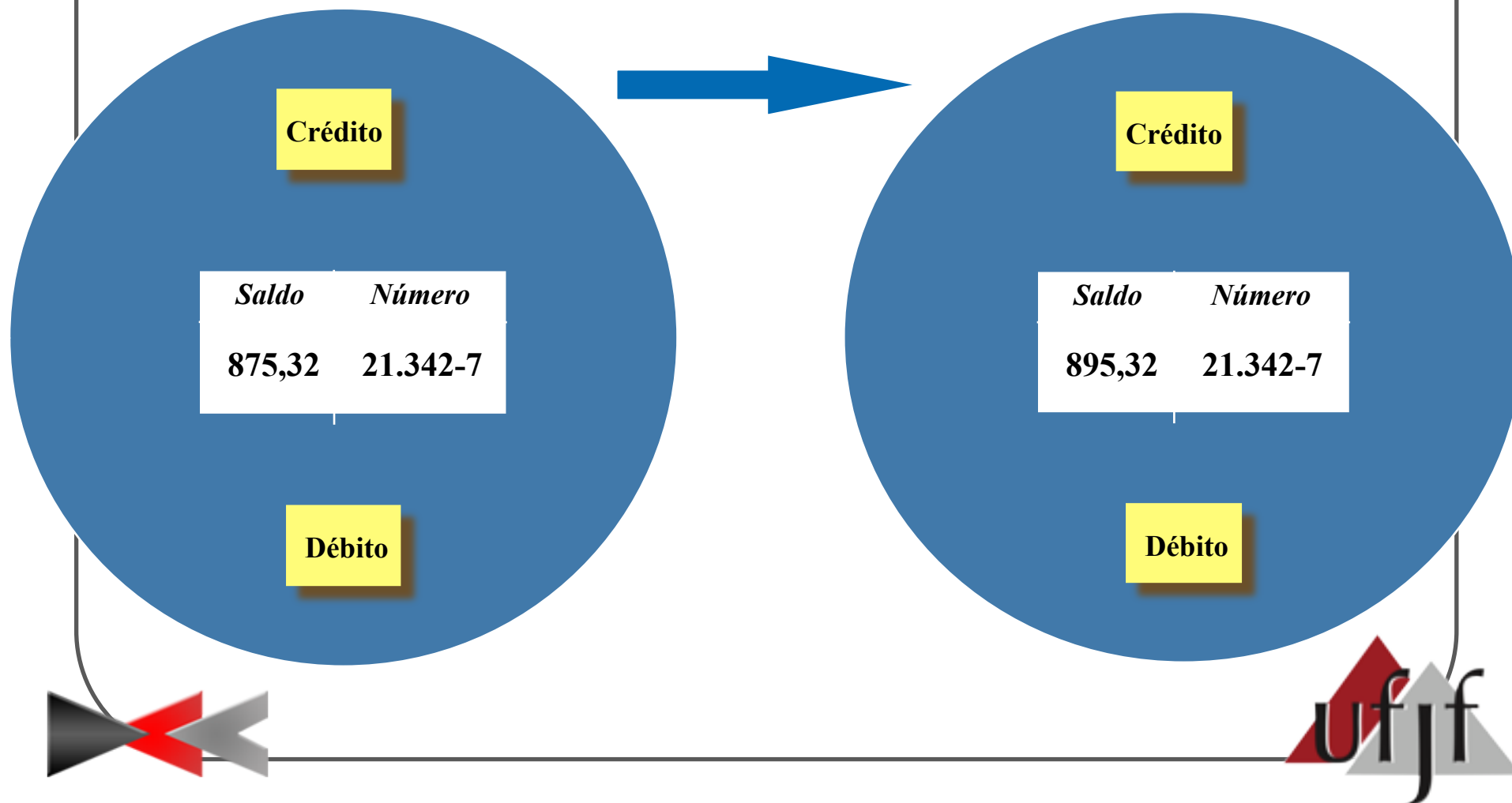
Crédito

<i>Saldo</i>	<i>Número</i>
875,32	21.342-7

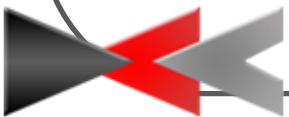
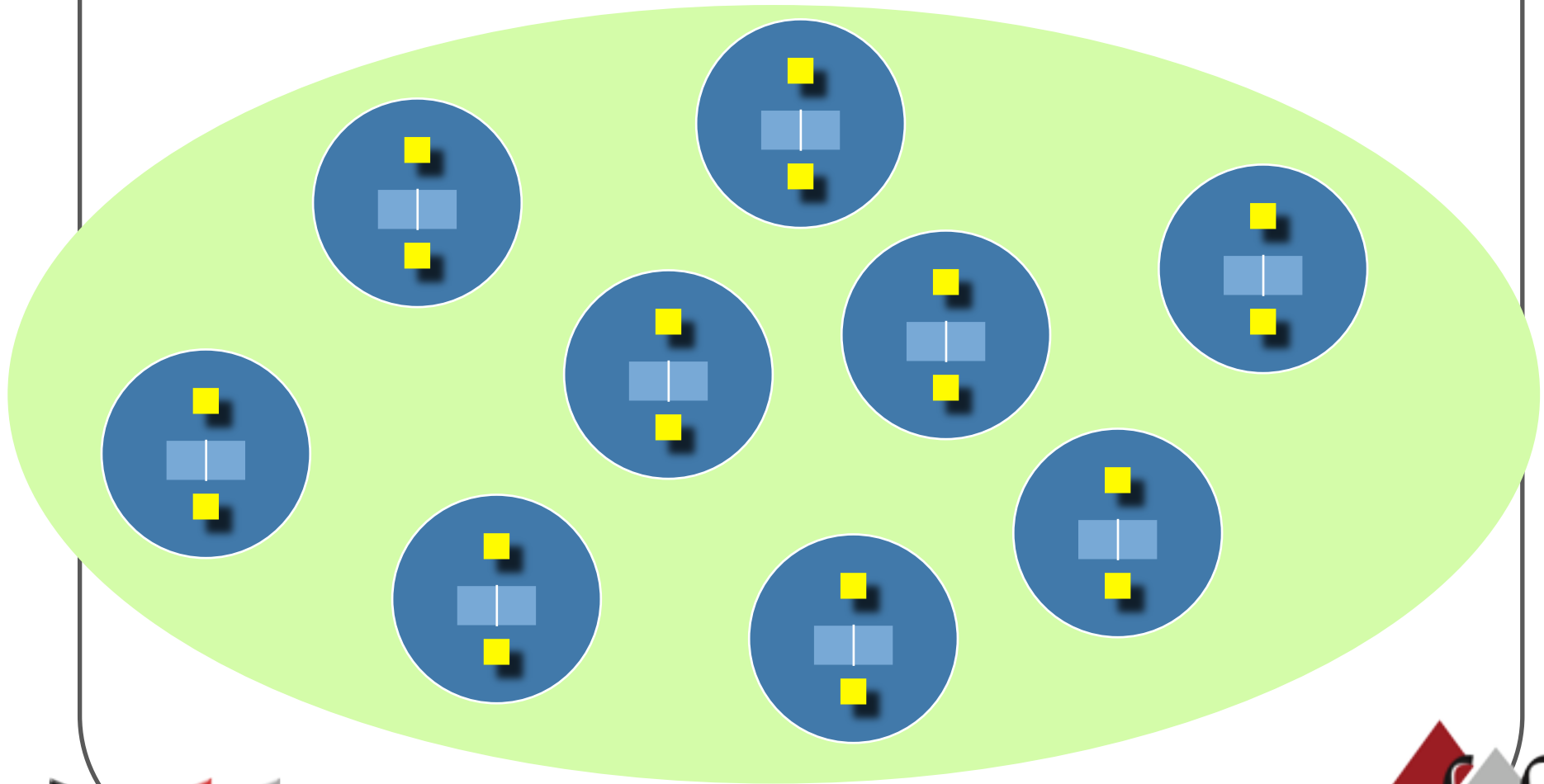
Débito



Estados do Objeto Conta



Classe de Contas Bancárias



Classes e Objetos

Objetos

métodos + atributos

estado encapsulado

Classes

agrupamento de objetos do mesmo tipo



Classe de Contas em Java

```
class Conta {  
    String  numero;  
    double  saldo;  
    void credito (double valor) {  
        saldo = saldo + valor;}  
    void debito (double valor) {  
        saldo = saldo - valor;}  
}
```



Criando Instâncias (Objetos)

Forma geral da declaração e criação de objetos

- Declaração

`<id-classe> <lista-id>;`

exemplo: Conta `conta1, conta2;`

- Criação

`<id> = new <id-classe> (<args>);`

exemplo: `conta1 = new Conta ();`

- Combinando declaração e criação

`<id-classe> <id> = new <id-classe> (<args>)`

exemplo: Conta `conta1 = new Conta ();`



Ex.: programa que cria e manipula 1 conta

```
class CriaConta {  
    /** Criando um objeto do tipo Conta */  
    public static void main (String [] args) {  
        Conta conta1 = new Conta ();  
        conta1.numero = "21.342-7"; // referencia a  
        atributos  
        conta1.saldo = 0;  
        conta1.credito (500.87); // referencia a metodos  
        conta1.debito (45.00);  
        System.out.println(conta1.saldo);  
    }  
}
```



Construtores

Além de atributos, classes podem ter **construtores** servem como interfaces para inicializar objetos possuem o mesmo nome das respectivas classes similares a métodos, mas não têm tipo de retorno pode haver mais de um por classe (**overloading**)

Ex.: `class Conta {`
 `String numero;`
 `double saldo;`
 `void credito (double valor) {saldo = saldo + valor;}`
 `void debito (double valor) {saldo = saldo - valor;}`
 `Conta (String n) {numero = n; saldo = 0;}`
 `}`



Construtores

Além de atributos, classes podem ter **construtores** servem como interfaces para inicializar objetos possuem o mesmo nome das respectivas classes similares a métodos, mas não têm tipo de retorno pode haver mais de um por classe (**overloading**)

Ex.: `class Conta {`
 `String numero;`
 `double saldo;`
 `void credito (double valor) {saldo = saldo + valor;}`
 `void debito (double valor) {saldo = saldo - valor;}`
 `Conta (String n) {numero = n; saldo = 0;}`
 `}`



Criando Objetos com Construtores

...

```
Conta conta1;
```

```
conta1 = new Conta("21.342-7");
```

```
conta1.credito(500.87);
```

```
conta1.debito(45.00);
```

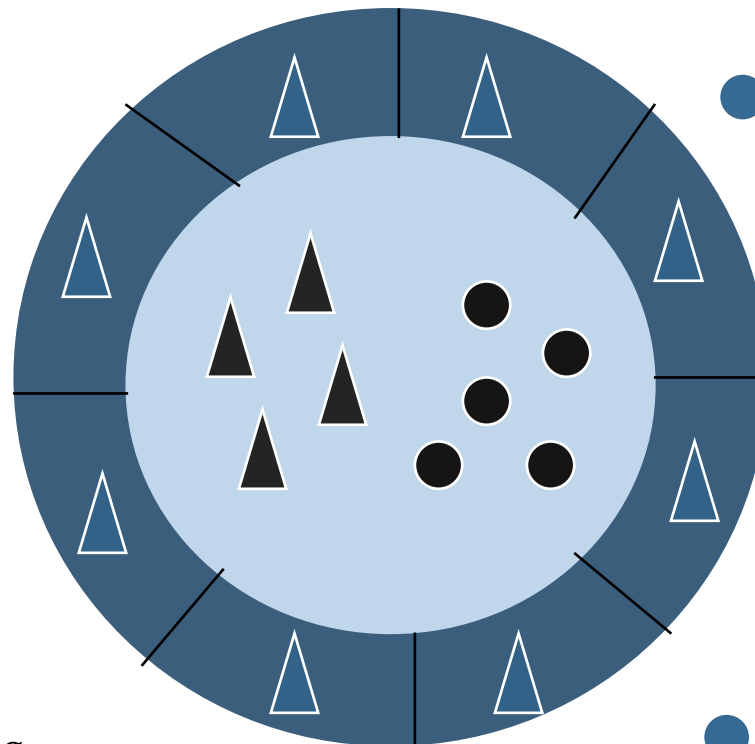
```
System.out.println(conta1.saldo);
```

....



Encapsulamento em uma classe

(Ref. Dominando o Java, Makron)



Métodos públicos

Métodos privados

Atributos públicos

Atributos privados



Controle de Acesso em Java

Normalmente, é conveniente proibir o acesso a certos

atributos (ou mesmo métodos). Os níveis de proteção extremos são:

- `public` - permite acesso a partir de qualquer classe
- `private` - permite acesso apenas na própria classe

Java oferece outros níveis de acesso que serão estudados posteriormente



Consequências de tornar um atributo privado

- Tentar acessar um componente privado (de fora da classe) resulta em erro de compilação
- Mas como torná-lo acessível apenas para consulta (leitura)?
- Isto é possível definindo-se um método que retorna o atributo (na própria classe onde o atributo se encontra)



Classe de Contas com Atributos Privados

```
class Conta {  
    private String  numero;  
    private double  saldo;  
    void credito (double valor) {  
        saldo = saldo + valor;}  
    void debito (double valor) {  
        saldo = saldo - valor;}  
    String numero() {return numero;}  
    double saldo() {return saldo;}  
    Conta (String n) {numero = n; saldo = 0;}
```



Exemplo Básico

- Protocolo de troca de mensagens
- O cliente envia uma mensagem ao servidor
- O servidor recebe a mensagem e envia o mesmo conteúdo ao cliente
- A mensagem recebida é apresentada



Stream Socket

- O cliente deve solicitar a conexão ao servidor
 - O servidor precisa criar um socket para receber a solicitação do cliente
- O cliente solicita a conexão através de:
 - Da criação de um socket TCP local
 - Especificação do IP e da porta do socket servidor
- Quando solicitado pelo cliente, o servidor cria um novo socket
 - Permite ao servidor se conectar a múltiplos clientes



TCP Cliente/Servidor

- Servidor é iniciado e preparado para aceitar novas conexões

Cliente

1. Cria um socket TCP
2. Comunica
3. Encerra a conexão

Servidor

1. Cria um socket TCP
2. Repetidamente:
 - a. Aceita uma nova conexão
 - b. Comunica
 - c. Encerra a conexão



Classes Java

- ServerSocket e Socket
 - O servidor mantém uma instância ServerSocket que permanece escutando as solicitações de conexão
 - Para cada conexão solicitada, o servidor instancia um Socket único
 - Assim, uma instância Socket deve ser mantida em cada lado da conexão (cliente e servidor)



TCP Servidor

1. Instanciar o ServerSocket em uma porta
2. Aceitar as conexões solicitadas, criando instâncias da classe Socket
3. Para cada instância da classe Socket, receber e enviar as mensagens.
4. Ao final da transmissão, fechar a conexão do cliente



TCP Cliente

1. Instanciar o Socket. O construtor solicita uma conexão para um endereço e porta específicos
2. Receber e enviar as mensagens
3. Fechar a conexão



TCP Cliente/Servidor

```
ServerSocket servSock = new ServerSocket(servPort);
```

Cliente

1. Cria um socket TCP
2. Comunica
3. Encerra a conexão

Servidor

1. Cria um socket TCP
2. Repetidamente:
 - a. Aceita uma nova conexão
 - b. Comunica
 - c. Encerra a conexão



TCP Cliente/Servidor

```
for (;;) {  
    Socket clntSock = servSock.accept();
```

Cliente

1. Cria um socket TCP
2. Comunica
3. Encerra a conexão

Servidor

1. Cria um socket TCP
2. **Repetidamente:**
 - a. **Aceita uma nova conexão**
 - b. Comunica
 - c. Encerra a conexão



TCP Cliente/Servidor

```
Socket socket = new Socket(server, servPort);
```

Cliente

1. Cria um socket TCP
2. Comunica
3. Encerra a conexão

Servidor

1. Cria um socket TCP
2. Repetidamente:
 - a. Aceita uma nova conexão
 - b. Comunica
 - c. Encerra a conexão



TCP Cliente/Servidor

```
OutputStream out = socket.getOutputStream();  
out.write(byteBuffer);
```

Cliente

1. Cria um socket TCP
2. **Comunica**
3. Encerra a conexão

Servidor

1. Cria um socket TCP
2. **Repetidamente:**
 - a. **Aceita uma nova conexão**
 - b. Comunica
 - c. Encerra a conexão



TCP Cliente/Servidor

```
InputStream in = clntSock.getInputStream();  
recvMsgSize = in.read(byteBuffer);
```

Cliente

1. Cria um socket TCP
2. **Comunica**
3. Encerra a conexão

Servidor

1. Cria um socket TCP
2. Repetidamente:
 - a. Aceita uma nova conexão
 - b. **Comunica**
 - c. Encerra a conexão



TCP Cliente/Servidor

`close(sock);`

`close(clntSocket)`

Cliente

1. Cria um socket TCP
2. Comunica
3. **Encerra a conexão**

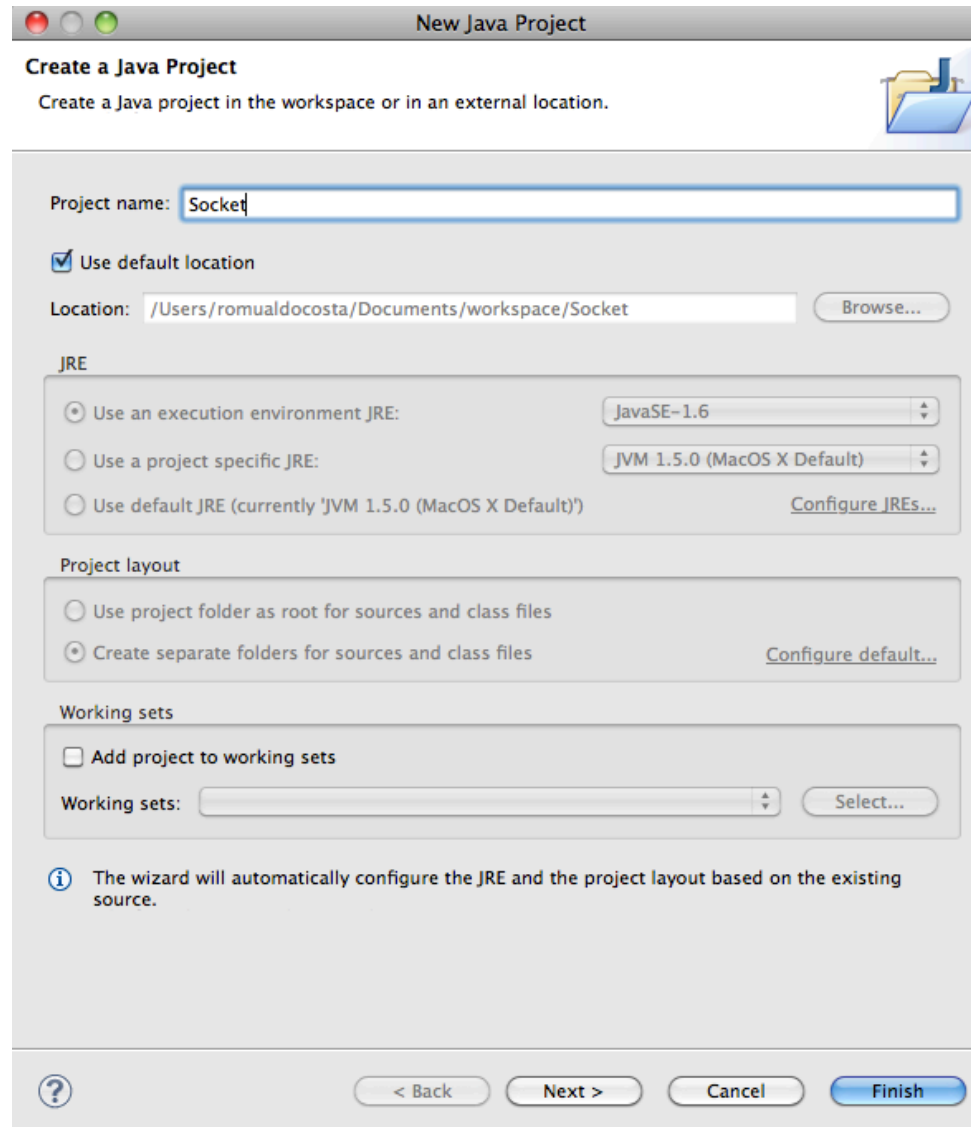
Servidor

1. Cria um socket TCP
2. Repetidamente:
 - a. Aceita uma nova conexão
 - b. Comunica
 - c. **Encerra a conexão**



Començando

- File
 - New
 - Java Project
 - Socket
- No mínimo
javaSE-1.6



Java

- Inúmeros pacotes, classes e métodos
- Consultar
 - <http://docs.oracle.com/javase/7/docs/api/>
- A cada versão novas adições
 - Impossível conhecer todas
 - 1.3 = Socket = 23 métodos
 - 7 = Socket = 42 métodos



docs.oracle.com/javase/7/docs/api/

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.nio.channels

Class SocketChannel

java.lang.Object
 java.nio.channels.spi.AbstractInterruptibleChannel
 java.nio.channels.SelectableChannel
 java.nio.channels.spi.AbstractSelectableChannel
 java.nio.channels.SocketChannel

All Implemented Interfaces:

Closeable, AutoCloseable, ByteChannel, Channel, GatheringByteChannel, InterruptibleChannel, NetworkChannel, ReadableByteChannel, ScatteringByteChannel, WritableByteChannel

```
public abstract class SocketChannel
extends AbstractSelectableChannel
implements ByteChannel, ScatteringByteChannel, GatheringByteChannel, NetworkChannel
```

A selectable channel for stream-oriented connecting sockets.

A socket channel is created by invoking one of the [open](#) methods of this class. It is not possible to create a channel for an arbitrary, pre-existing socket. A newly-created socket channel is open but not yet connected. An attempt to invoke an I/O operation upon an unconnected channel will cause a [NotYetConnectedException](#) to be thrown. A socket channel can be connected by invoking its [connect](#) method; once connected, a socket channel remains connected until it is closed. Whether or not a socket channel is connected may be determined by invoking its [isConnected](#) method.

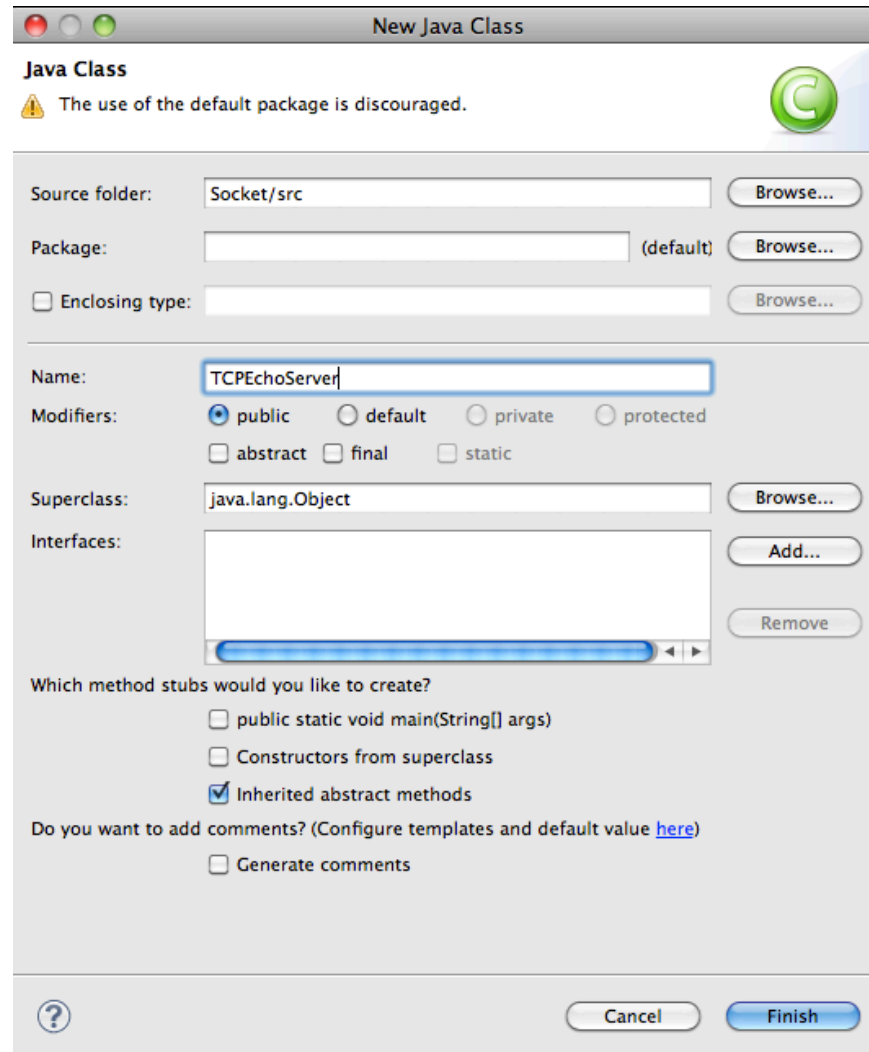
Socket channels support *non-blocking connection*: A socket channel may be created and the process of establishing the link to the remote socket may be initiated via the [connect](#) method for later completion by the [finishConnect](#) method. Whether or not a connection operation is in progress may be determined by invoking the [isConnectionPending](#) method.

Socket channels support *asynchronous shutdown*, which is similar to the asynchronous close operation specified in the [Channel](#) class. If the input side of a socket is shut down by one thread while another thread is blocked in a read operation on the socket's channel, then the read operation in the blocked thread will complete without reading any bytes and will return -1. If the output side of a socket is shut down by one thread while another thread is blocked in a write operation on the socket's channel, then the blocked thread will receive an [AsynchronousCloseException](#).



TCPEchoServer

- Selezione o Progetto
 - File
 - New
 - Class
 - TCPEchoServer

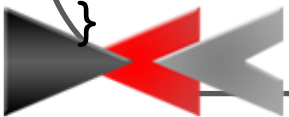


ServerSocket

```
import java.net.*;    // Socket, ServerSocket
import java.io.*;     // Input/OutputStream

public class TCPEchoServer {
    private static final int BUFSIZE = 32;

    public static void main(String[] args) throws
        IOException{
        int servPort = Integer.parseInt(args[0]);
        ServerSocket servSock = new
        ServerSocket(servPort);
    }
}
```



Socket

```
int recvMsgSize;    // Tamanho da msg
byte[] byteBuffer = new byte[BUFSIZE]; // Buffer de recebimento

for (;;) {
    // Espera as solicitações dos clientes
    Socket clntSock = servSock.accept();

    System.out.println("Controlando cliente " +
        clntSock.getInetAddress().getHostAddress() + " na porta " +
        clntSock.getPort());

    InputStream in = clntSock.getInputStream();
    OutputStream out = clntSock.getOutputStream();

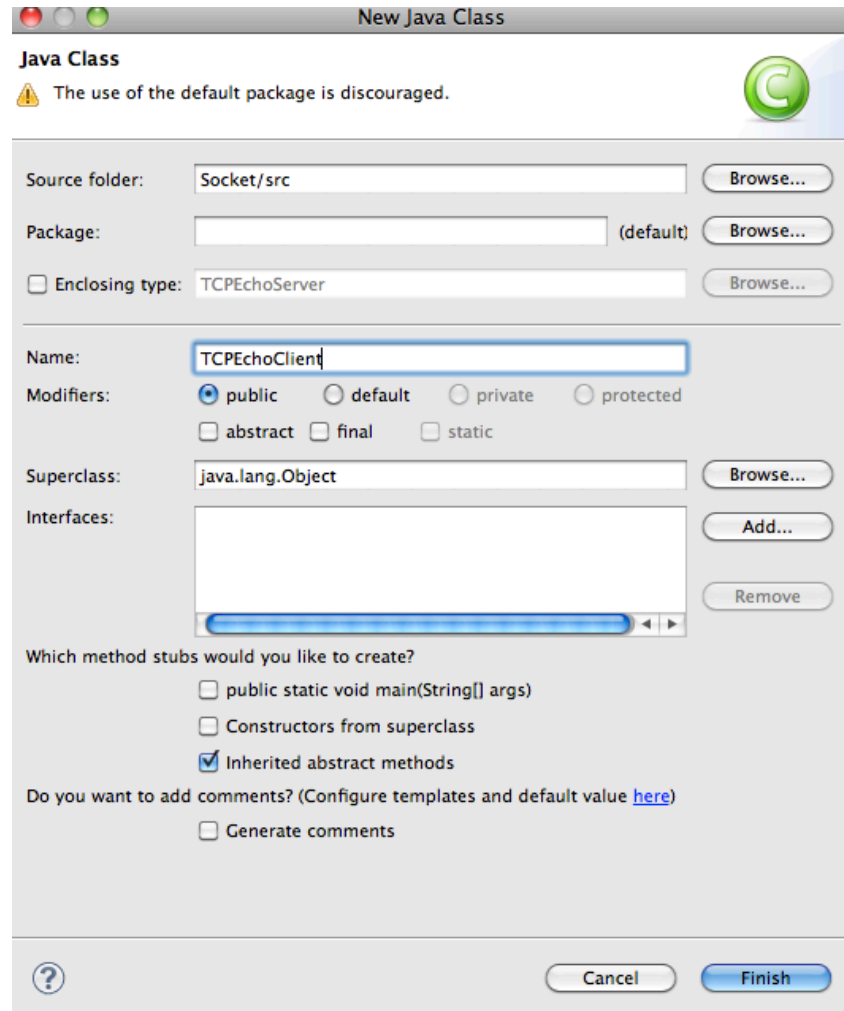
    while ((recvMsgSize = in.read(byteBuffer)) != -1)
        out.write(byteBuffer, 0, recvMsgSize);

    clntSock.close();
}
```



TCPEchoClient

- Seleccione o Projeto
 - File
 - New
 - Class
 - TCPEchoClient



Cliente

```
import java.net.*; // Socket
import java.io.*;  // Input/OutputStream

public class TCPEchoClient {

    public static void main(String[] args) throws IOException {

        String server = args[0]; // Nome do Servidor ou Endereço IP
        // Converte a entrada de String para bytes
        byte[] byteBuffer = args[1].getBytes();
        int servPort = Integer.parseInt(args[2]);

        // Cria o socket conectado ao servidor e a porta específica
        Socket socket = new Socket(server, servPort);
        System.out.println("Conectado ao servidor...enviando texto");
```



Cliente

```
InputStream in = socket.getInputStream();
OutputStream out = socket.getOutputStream();

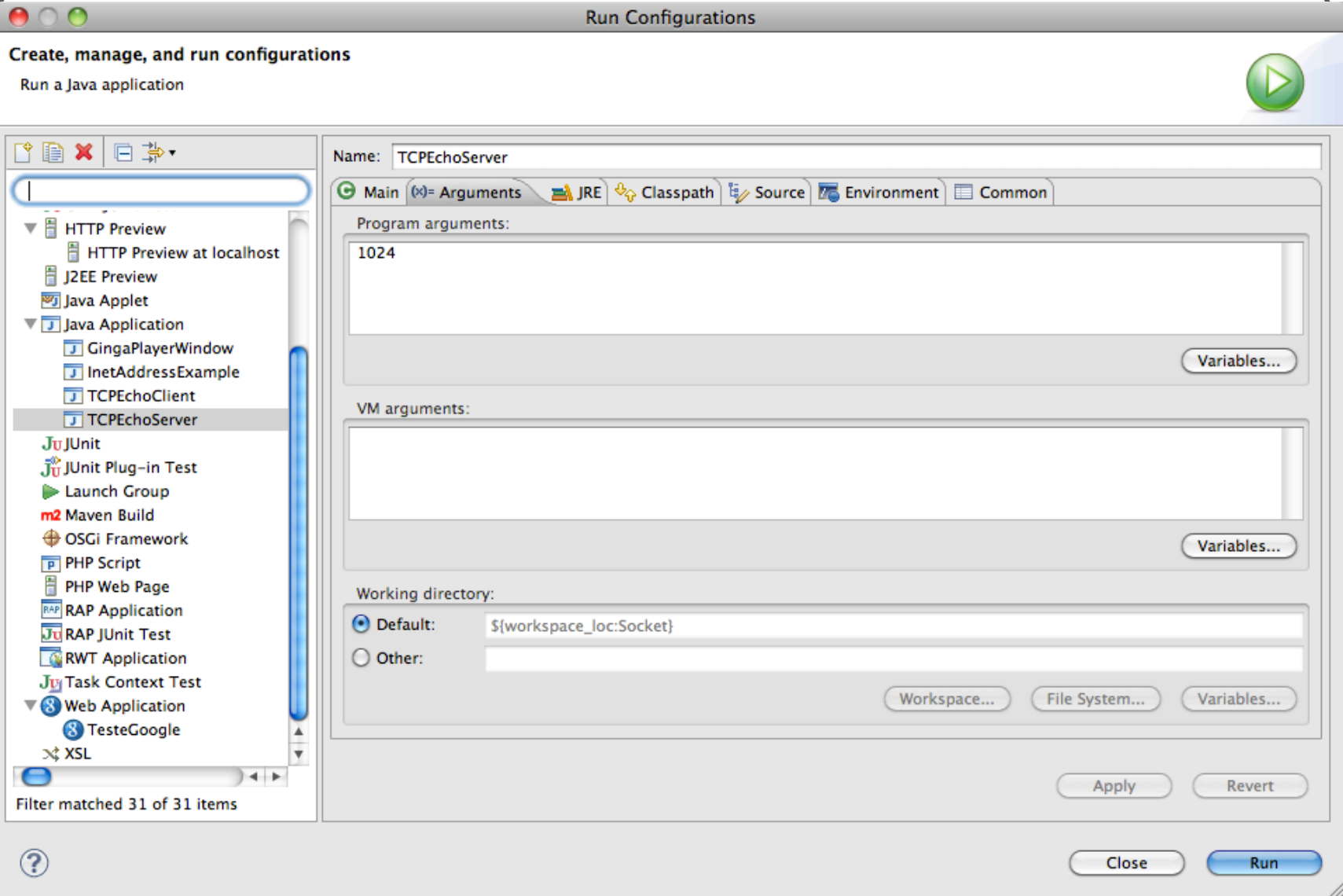
out.write(byteBuffer); // Envia o texto ao servidor
// Recebe o mesmo texto do servidor
int totalBytesRcvd = 0;
int bytesRcvd;
while (totalBytesRcvd < byteBuffer.length) {
    if ((bytesRcvd = in.read(byteBuffer, totalBytesRcvd,
        byteBuffer.length - totalBytesRcvd)) == -1) throw
new SocketException("Conexão encerrada");
    totalBytesRcvd += bytesRcvd;
}
System.out.println("Recebido: " + new String(byteBuffer));
socket.close(); // Encerra o socket
}
}
```



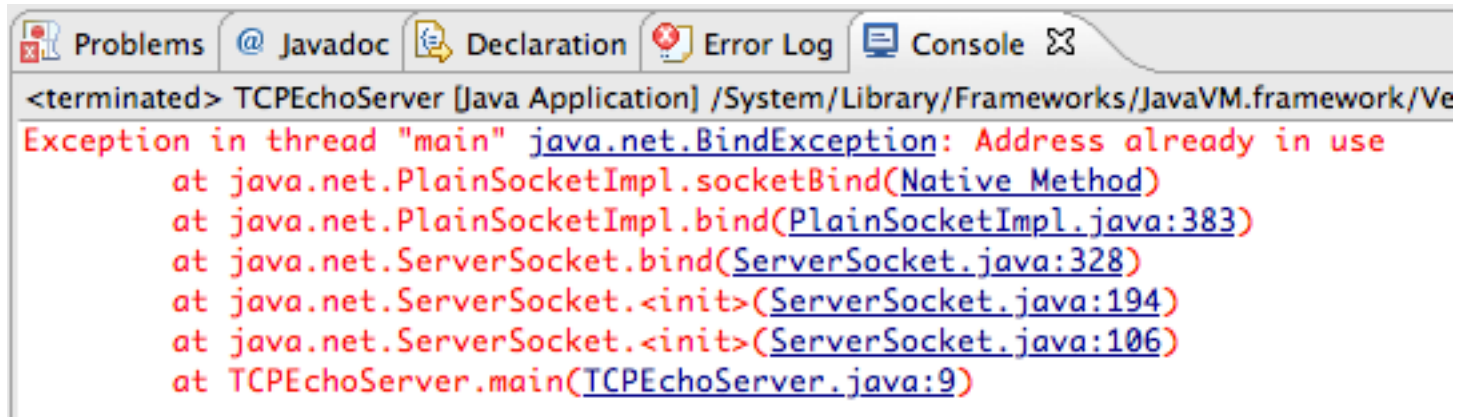
Execução

- Run
 - Configurations
 - Arguments
- Servidor
 - Porta
- Cliente
 - Servidor
 - Texto
 - Porta



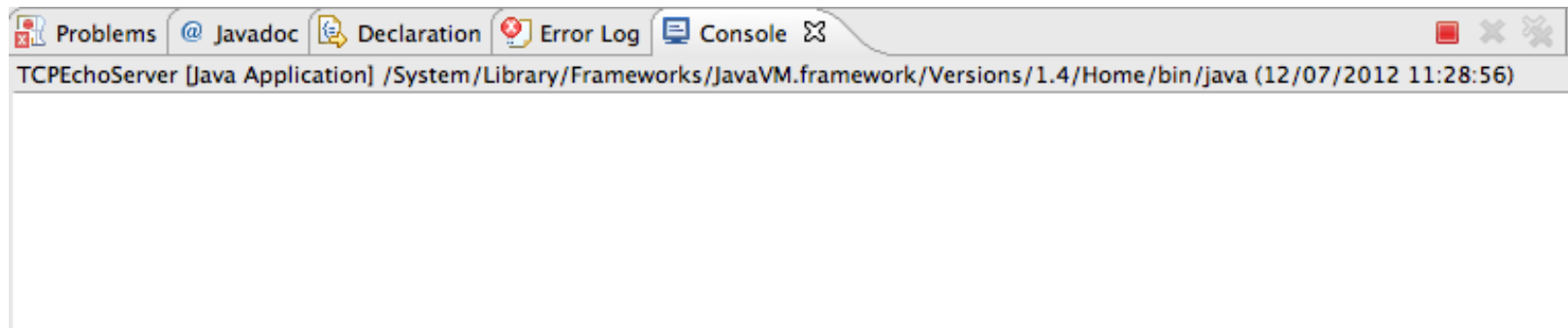


Resultado



The screenshot shows an IDE console window with tabs for Problems, Javadoc, Declaration, Error Log, and Console. The Console tab is active, displaying the following text:

```
<terminated> TCPEchoServer [Java Application] /System/Library/Frameworks/JavaVM.framework/Ve  
Exception in thread "main" java.net.BindException: Address already in use  
    at java.net.PlainSocketImpl.socketBind(Native Method)  
    at java.net.PlainSocketImpl.bind(PlainSocketImpl.java:383)  
    at java.net.ServerSocket.bind(ServerSocket.java:328)  
    at java.net.ServerSocket.<init>(ServerSocket.java:194)  
    at java.net.ServerSocket.<init>(ServerSocket.java:106)  
    at TCPEchoServer.main(TCPEchoServer.java:9)
```

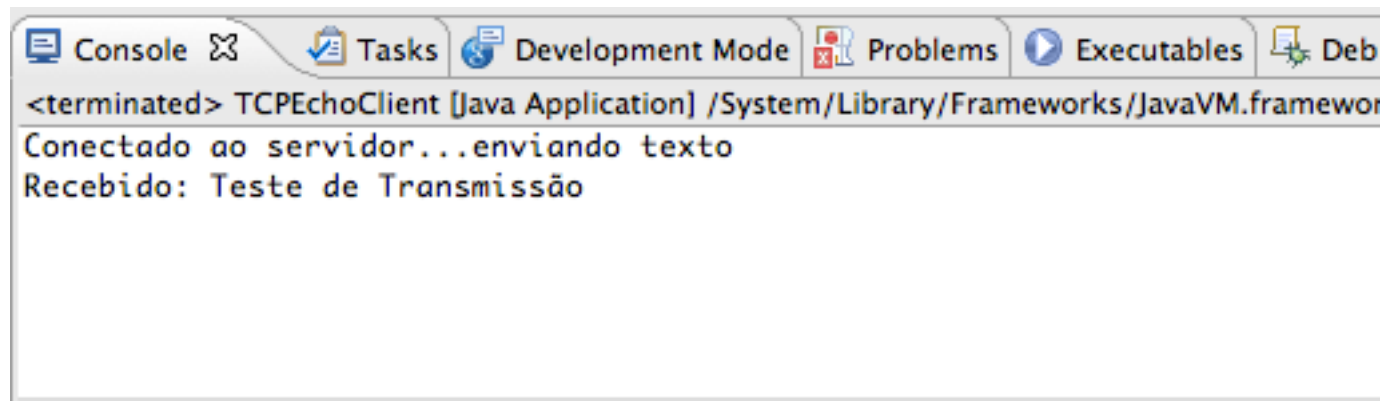


The screenshot shows an IDE console window with tabs for Problems, Javadoc, Declaration, Error Log, and Console. The Console tab is active, displaying the following text:

```
TCPEchoServer [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.4/Home/bin/java (12/07/2012 11:28:56)
```

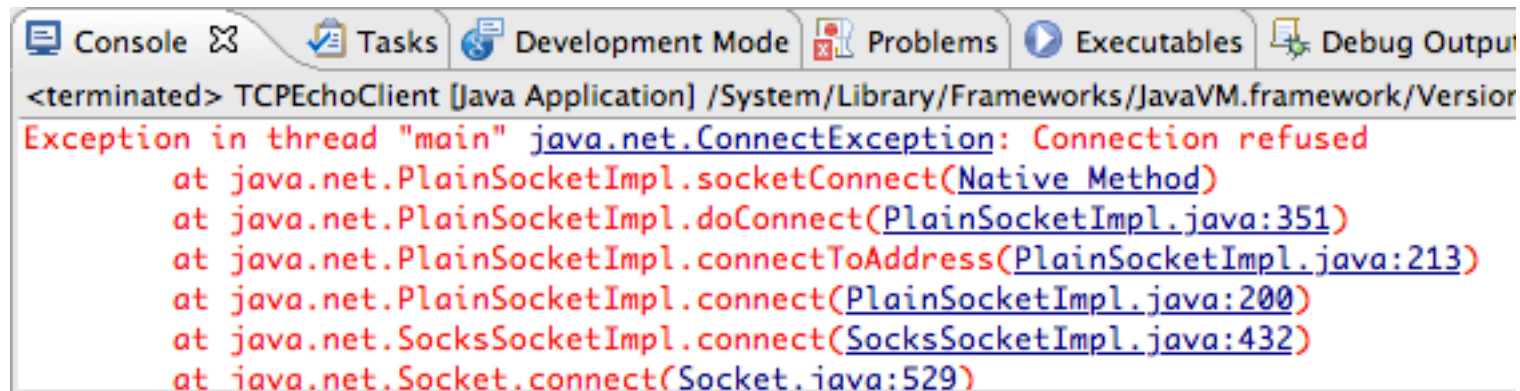


Resultado



The screenshot shows an IDE console window with tabs for Console, Tasks, Development Mode, Problems, Executables, and Debug. The console output for a Java application named TCPEchoClient shows a successful connection to a server and the receipt of a message.

```
<terminated> TCPEchoClient [Java Application] /System/Library/Frameworks/JavaVM.framework  
Conectado ao servidor...enviando texto  
Recebido: Teste de Transmissão
```

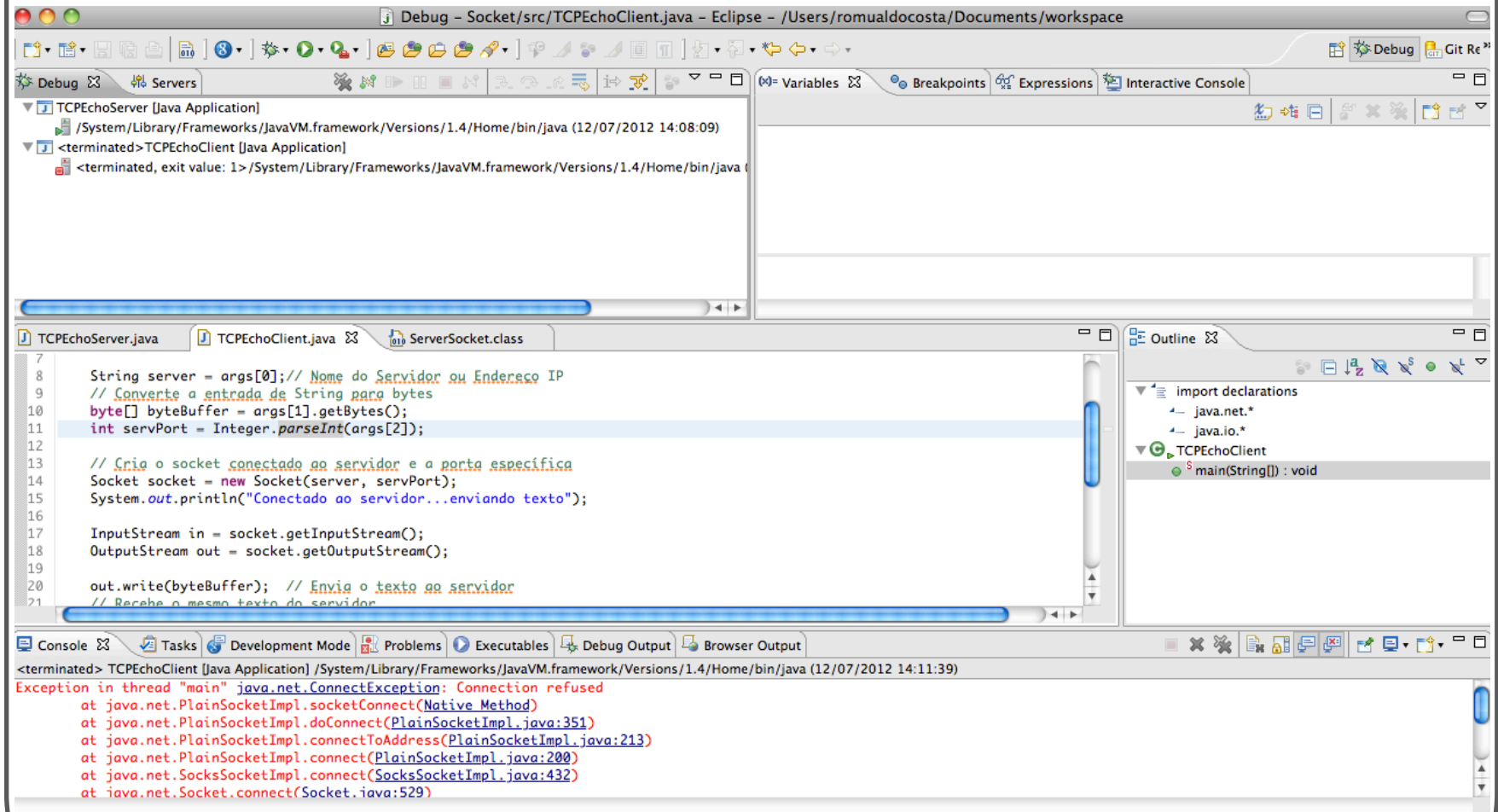


The screenshot shows an IDE console window with tabs for Console, Tasks, Development Mode, Problems, Executables, and Debug Output. The console output for a Java application named TCPEchoClient shows a 'Connection refused' exception in the main thread.

```
<terminated> TCPEchoClient [Java Application] /System/Library/Frameworks/JavaVM.framework/Version  
Exception in thread "main" java.net.ConnectException: Connection refused  
    at java.net.PlainSocketImpl.socketConnect(Native Method)  
    at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:351)  
    at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:213)  
    at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:200)  
    at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:432)  
    at java.net.Socket.connect(Socket.java:529)
```



Dúvidas - Debug



The screenshot shows the Eclipse IDE interface during a debug session. The top toolbar includes buttons for Debug, Servers, Variables, Breakpoints, Expressions, and Interactive Console. The left sidebar shows the project structure with 'TCPEchoServer' and 'TCPEchoClient' applications. The main editor displays the source code of 'TCPEchoClient.java' with the following content:

```
7
8 String server = args[0]; // Nome do Servidor ou Endereço IP
9 // Converte a entrada de String para bytes
10 byte[] byteBuffer = args[1].getBytes();
11 int servPort = Integer.parseInt(args[2]);
12
13 // Cria o socket conectado ao servidor e a porta específica
14 Socket socket = new Socket(server, servPort);
15 System.out.println("Conectado ao servidor...enviando texto");
16
17 InputStream in = socket.getInputStream();
18 OutputStream out = socket.getOutputStream();
19
20 out.write(byteBuffer); // Envia o texto ao servidor
21 // Recebe o mesmo texto do servidor
```

The right sidebar shows the 'Outline' view with the following structure:

- import declarations
 - java.net.*
 - java.io.*
- TCPEchoClient
 - main(String[]) : void

The bottom console shows the following output:

```
<terminated> TCPEchoClient [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.4/Home/bin/java (12/07/2012 14:11:39)
Exception in thread "main" java.net.ConnectException: Connection refused
    at java.net.PlainSocketImpl.socketConnect(Native Method)
    at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:351)
    at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:213)
    at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:200)
    at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:432)
    at java.net.Socket.connect(Socket.java:529)
```



Entrada e Saída

- Informação enviada pela rede pode ser tratada como um arquivo local
 - I/O em Java = manipulação se Streams
 - Sequência ordenada de bytes
- InputStream
 - Leitura de bytes
- OutputStream
 - Escrita de bytes



OutputStream

- Métodos Básicos
 - void write(byte[] data)
 - void write(byte[] data, int offset, int length)
 - void flush()
 - void close()



InputStream

- Métodos Básicos
 - `int read(byte[] data)`
 - `int read(byte[] data, int offset, int length)`
 - `int available()`
 - `void close()`



InputStream

- Método read() bloqueia a execução do programa até que um byte esteja disponível
- Entrada e saída tendem a consumir muito tempo
- Pode ser recomendável que sua execução seja realizada através de processos leves (threads)



Socket

- Criação
 - `Socket(InetAddress remoteAddr, int remotePort)`
 - `Socket(String remoteHost, int remotePort)`
 - `Socket(InetAddress remoteAddr, int remotePort, InetAddress localAddr, int localPort)`
 - `Socket(String remoteHost, int remotePort, InetAddress localAddr, int localPort)`
 - `Socket()`



Socket

- Métodos Básicos
 - void connect(SocketAddress destination)
 - void connect(SocketAddress destination, int timeout)
 - InputStream getInputStream()
 - OutputStream getOutputStream()
 - void close()
 - void shutdownInput()
 - void shutdownOutput()



Socket

- Atributos Básicos
 - InetAddress getInetAddress()
 - int getPort()
 - InetAddress getLocalAddress()
 - int getLocalPort()
 - SocketAddress getRemoteSocketAddress()
 - SocketAddress getLocalSocketAddress()



InetSocketAddress

- Criação, Métodos e Atributos Básicos:
 - InetSocketAddress(InetAddress addr, int port)
 - InetSocketAddress(int port)
 - InetSocketAddress(String hostname, int port)
 - static InetSocketAddress createUnresolved(String host, int port)
 - boolean isUnresolved()
 - InetAddress getAddress()
 - int getPort()
 - String getHostName()
 - String toString()



ServerSocket

- Criação
 - `ServerSocket(int localPort)`
 - `ServerSocket(int localPort, int queueLimit)`
 - `ServerSocket(int localPort, int queueLimit, InetAddress localAddr)`
 - `ServerSocket()`



ServerSocket

- Métodos Básicos
 - void bind(int port)
 - void bind(int port, int queueLimit)
 - Socket accept()
 - void close()



ServerSocket

- Atributos Básicos
 - InetAddress getInetAddress()
 - SocketAddress getLocalSocketAddress()
 - int getLocalPort()



Estruturas

- InetAddress
 - Representa um endereço de rede
- Inet4Address
 - Representa um endereço IPv4
- Inet6Address
 - Representa um endereço IPv6
- NetworkInterface
 - Interface de rede



Exemplo das Estruturas

- NetworkInterface e InetAddress
 - Identificar o endereço IP da máquina
- Selecione o projeto:
 - File
 - New
 - Class
 - InetAddressExample



InetAddress

```
import java.net.*;
import java.io.*;

public class InetAddressExample {

    public static void main(String[] args) throws
IOException {

        InetAddress address =
InetAddress.getLocalHost();
        System.out.println(address.getHostName());
        System.out.println(address.getHostAddress());
    }
}
```



Exercício

Defina uma classe para representar um candidato a uma eleição, com atributos para armazenar o nome do candidato e o número de votos, ambos privados. Defina um método de acesso para cada atributo, um método para incrementar o número de votos do candidato e um construtor para a classe que recebe como argumento o nome do candidato e inicializa o número de votos com zero.

Desenvolva um programa que cria um candidato com nome de sua escolha. Em seguida, deve ser lido um nome. Se o nome lido for o do seu candidato, incrementar o contador de votos. No final, imprimir o nome do candidato e o total de votos.

