

Comunicação entre processos

Fernando Afonso Santos

*UNIFEI
Campus Itabira*

Processos

- Processos e Programas
 - Um programa é uma entidade estática
 - Fornece as instruções necessárias para realizar tarefas
 - Não concorre por recursos do sistema
 - Um processo é um programa em execução
 - Caracteriza-se por uma entidade dinâmica, que muda de estados e concorre por recursos do sistema
 - Cada processo está associado a um Bloco de Controle de Processo (PCB), composto por informações como: contador de programa (PC), descritores de arquivo e endereços de memória

Processos

- Os processos executando no Sistema Operacional podem ser independentes ou cooperativos
 - Processos independentes não compartilham dados com outros processos, os cooperativos sim.
- Principais motivos para cooperação entre processos
 - Compartilhamento de informações
 - Velocidade de computação
 - Modularidade

Processos

- A cooperação requer que os processos **comuniquem entre si** e sincronizem suas ações
- Existem diferentes estratégias de comunicação entre processos - IPC (*Inter-Process Communication*)
 - A maneira mais comum para a comunicação entre processos é a troca de mensagens

Comunicação entre processos

- O mecanismo de troca de mensagens permite a processos locais ou remotos comunicarem
 - Para a comunicação existir deve haver entre eles um **canal de comunicação**
 - A implementação da maioria destes canais se baseia nas primitivas de mensagens **send(msg)** e **receive(msg)**

Comunicação entre processos

- Diferentes implementações são possíveis para a definição de canais de comunicação
 - Comunicação síncrona/assíncrona
 - Comunicação confiável/não-confiável
 - Comunicação orientada/não-orientada a conexão

Comunicação entre processos

- Diferentes implementações são possíveis para a definição de canais de comunicação
 - Comunicação síncrona/assíncrona
 - Comunicação confiável/não-confiável
 - Comunicação orientada/não-orientada a conexão
- Suponha que existam processos P e Q que desejam comunicar
 - P deseja enviar uma mensagem a Q

Comunicação entre processos

Comunicação síncrona/assíncrona

- Troca de mensagens síncrona
 - $\text{send}(msg, Q)$: P bloqueia até Q receber a mensagem
 - $\text{receive}(msg, P)$: Q bloqueia até receber a mensagem de P
 - Conhecida como comunicação *rendezvous*
- Troca de mensagens assíncrona
 - $\text{send}(msg, Q)$: P envia e continua sua execução, independente se Q recebeu ou não a mensagem
 - $\text{receive}(msg, P)$: Q recebe a mensagem de P ou nada, e continua sua execução

Comunicação entre processos

Comunicação confiável/não-confiável

- Canal Confiável
 - Estabelecido um canal de comunicação [send(*msg*,Q), receive(*msg*,P)], garante-se que Q receberá *msg* de P
 - Existe um *overhead* para assegurar a confiabilidade
- Canal Não-Confiável
 - Ao enviar uma mensagem entre dois processos, não há garantia que esta mensagem será entregue
 - Sem *overhead* de confiabilidade

Comunicação entre processos

Comunicação orientada/não-orientada a conexão

- Canal orientado a conexão
 - Antes de dois processos comunicarem, o canal é estabelecido e só deixa de existir quando finalizado por algum dos processos
 - Mesmo se ocioso, o canal continua ativo
- Canal não-orientado a conexão
 - O canal entre dois processos é estabelecido apenas no envio/recebimento de uma mensagem
 - Ao concluir o envio/recebimento, o canal deixa de existir

Sockets

- Um socket representa a extremidade de um canal de comunicação
 - Tendo-se dois ou mais sockets corretamente ‘conectados’ é possível estabelecer um canal de comunicação
- Na comunicação entre processos
 - É possível vincular processos a sockets e estabelecer um canal de comunicação
 - Deve haver um suporte da linguagem de programação para utilizar tal estratégia

Sockets TCP

- Aplica-se especificamente para estabelecer um canal de comunicação em redes TCP/IP
- Utiliza o conceito de portas
 - Cada endereço IP tem 65536 portas
 - Um socket TCP vincula o processo a uma porta, permitindo a comunicação entre processos
- O termo TCP (*Transmission Control Protocol*) vem devido à implementação de um canal confiável e orientado a conexão
 - A comunicação é feita por fluxos de dados

Sockets TCP

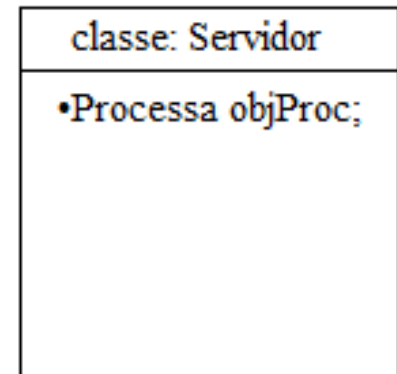
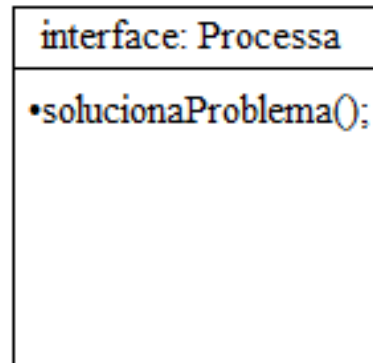
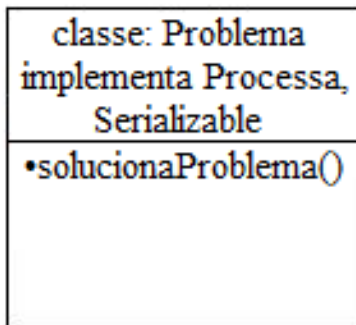
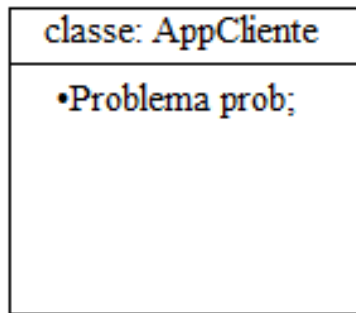
Exemplo de aplicação

- Uma empresa lida com problemas de elevada complexidade computacional
- Após uma pesquisa resolveu implantar um servidor de processamento, ao invés de aumentar a capacidade das estações
- Um processo no servidor recebe requisições, as processa e retorna o resultado à estação cliente
- A comunicação é realizada utilizando sockets TCP
- Implementação em Java

Sockets TCP

Exemplo de aplicação

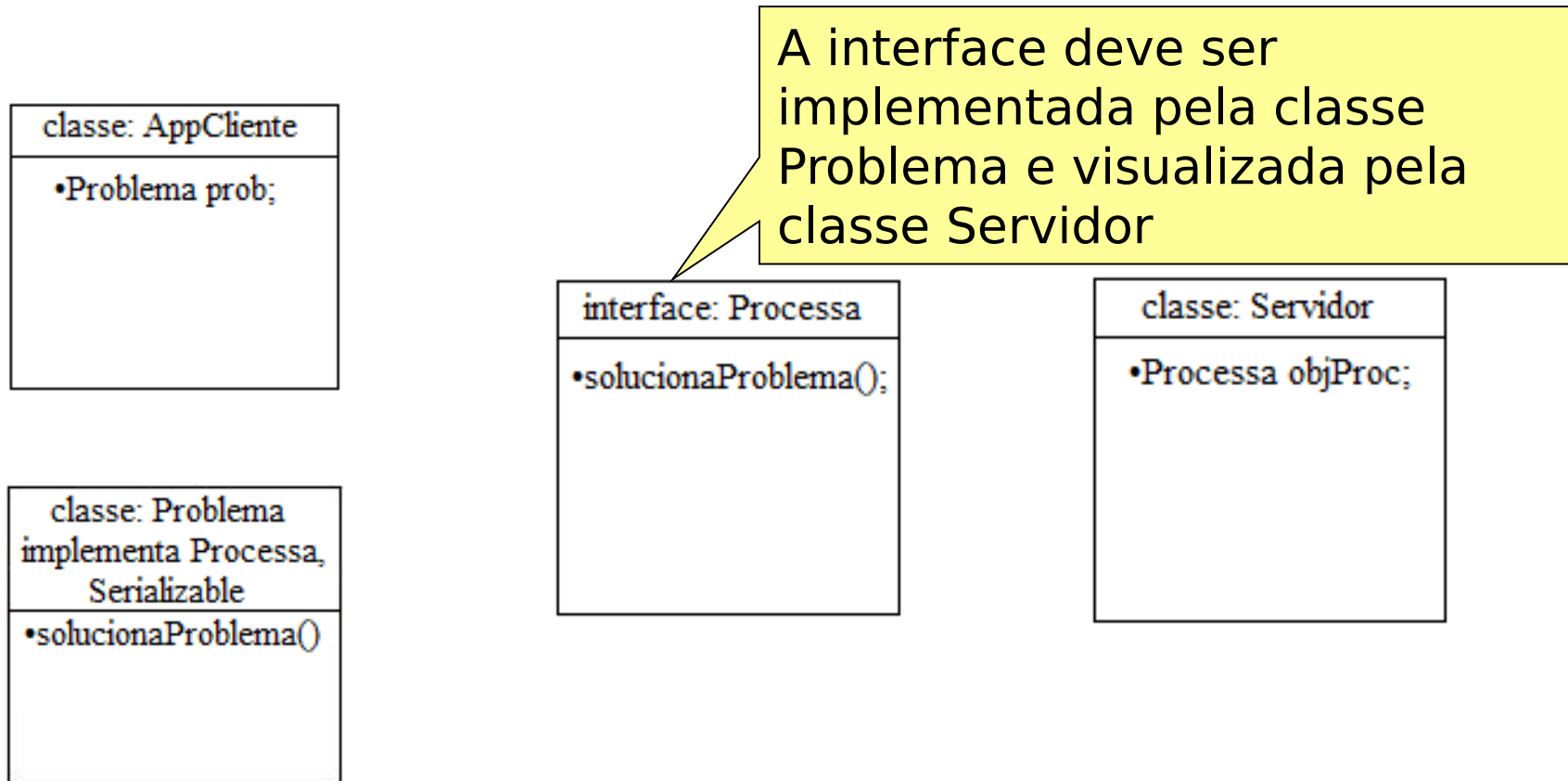
- Classes/Interfaces envolvidas na implementação



Sockets TCP

Exemplo de aplicação

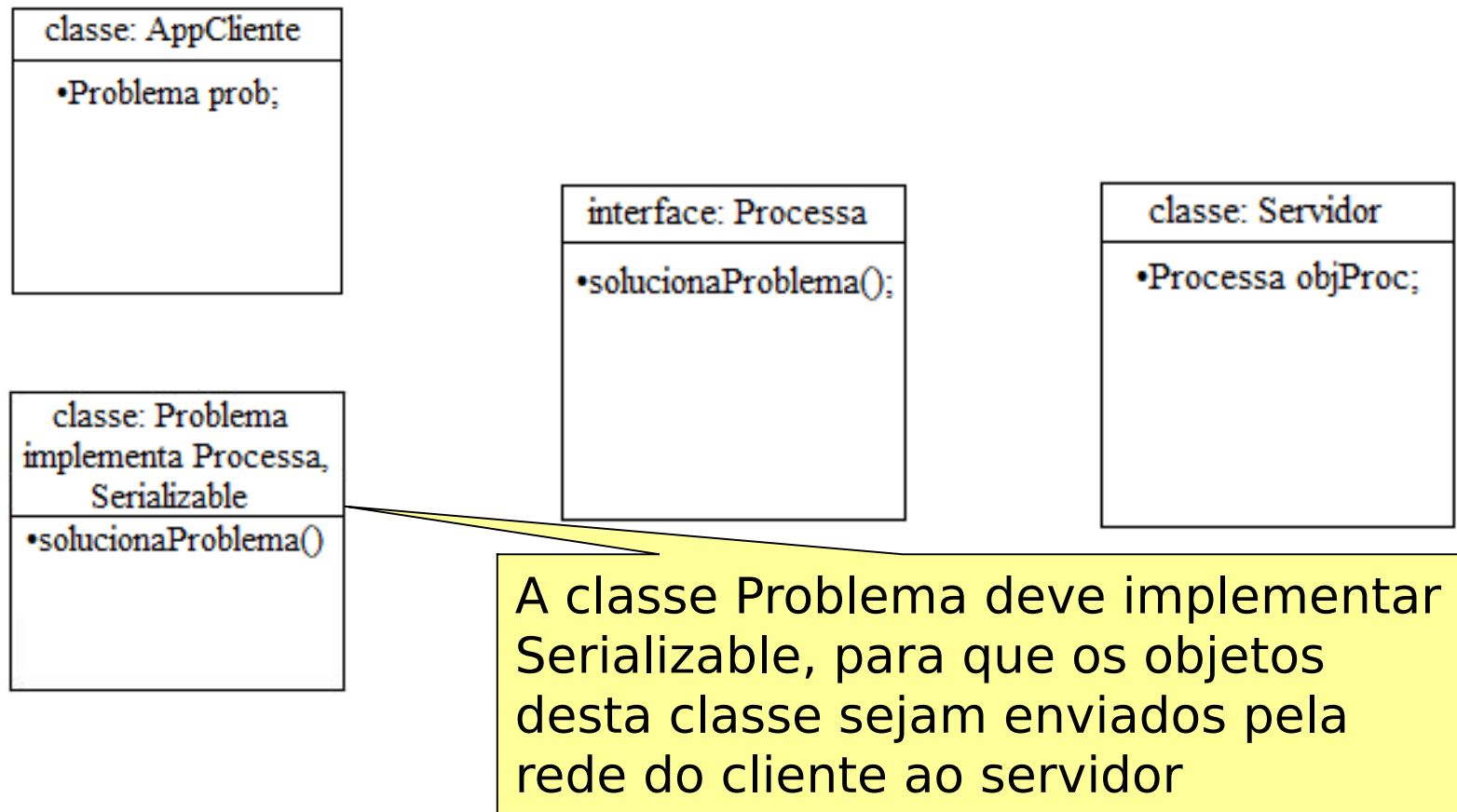
- Classes/Interfaces envolvidas na implementação



Sockets TCP

Exemplo de aplicação

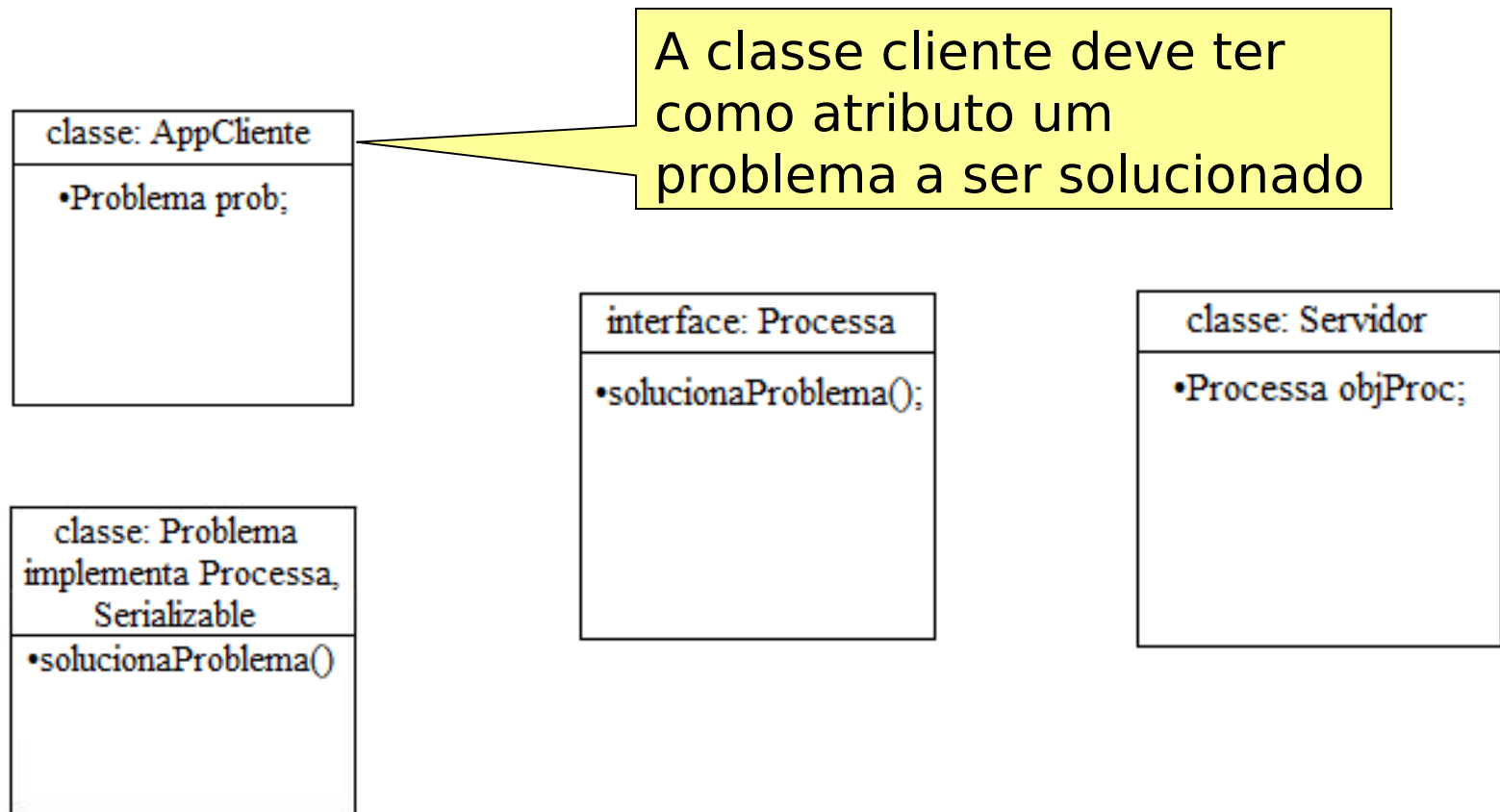
- Classes/Interfaces envolvidas na implementação



Sockets TCP

Exemplo de aplicação

- Classes/Interfaces envolvidas na implementação

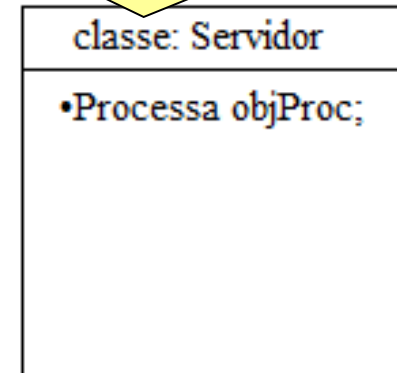
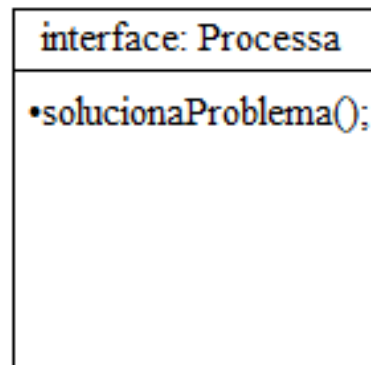
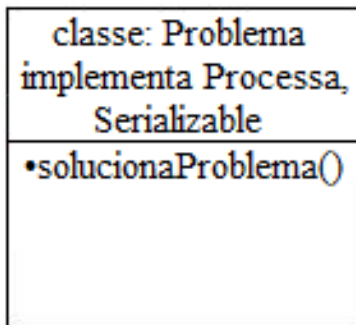
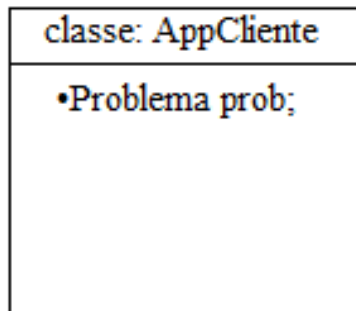


Sockets TCP

Exemplo de aplicação

- Classes/Interfaces envolvidas na implementação

A classe servidor tem como atributo um elemento da interface Processa, que permitirá invocar o método processaObjeto(), mesmo sem saber qual é este objeto



Sockets TCP

Exemplo de aplicação

```
public class Servidor {
    private ServerSocket servidor;
    private Socket conexao;
    private ObjectInputStream input;
    private ObjectOutputStream output;
    private Processa objProc;

    public void ativaServer(){
        servidor = new ServerSocket(56789, 10);

        while(true)
            conexao = servidor.accept();

            output = new ObjectOutputStream(conexao.getOutputStream());
            input = new ObjectInputStream(conexao.getInputStream());

            objProc = (Processa) input.readObject();
            objProc.processaObjeto();
            output.writeObject(objProc);
        }
    }
}
```

Sockets TCP

Exemplo de aplicação

```
public class Servidor {  
    private ServerSocket servidor;  
    private Socket conexao;  
    private ObjectInputStream input;  
    private ObjectOutputStream output;  
    private Processa objProc;
```

Vincula-se o processo servidor à porta 56789. As requisições enviadas ao endereço “IP servidor : 56789” serão atendidas por este processo

```
    public void ativaServer(){  
        servidor = new ServerSocket(56789, 10);
```

```
        while(true)  
            conexao = servidor.accept();
```

```
        output = new ObjectOutputStream(conexao.getOutputStream());  
        input = new ObjectInputStream(conexao.getInputStream());
```

```
        objProc = (Processa) input.readObject();  
        objProc.processaObjeto();  
        output.writeObject(objProc);
```

```
    }  
}  
}
```

Sockets TCP

Exemplo de aplicação

```
public class Servidor {  
    private ServerSocket servidor;  
    private Socket conexao;  
    private ObjectInputStream input;  
    private ObjectOutputStream output;  
    private Processa objProc;  
  
    public void ativaServer(){  
        servidor = new ServerSocket(56789, 10);  
  
        while(true)  
            conexao = servidor.accept();  
  
        output = new ObjectOutputStream(conexao.getOutputStream());  
        input = new ObjectInputStream(conexao.getInputStream());  
  
        objProc = (Processa) input.readObject();  
        objProc.processaObjeto();  
        output.writeObject(objProc);  
    }  
}
```

O servidor entra em espera, até que existam requisições para novas conexões. Ao aceitar uma conexão, esta permanece ativa até que um dos lados a interrompa

Sockets TCP

Exemplo de aplicação

```
public class Servidor {  
    private ServerSocket servidor;  
    private Socket conexao;  
    private ObjectInputStream input;  
    private ObjectOutputStream output;  
    private Processa objProc;  
  
    public void ativaServer(){  
        servidor = new ServerSocket(56789, 10);  
  
        while(true)  
            conexao = servidor.accept();  
  
        output = new ObjectOutputStream(conexao.getOutputStream());  
        input = new ObjectInputStream(conexao.getInputStream());  
  
        objProc = (Processa) input.readObject();  
        objProc.processaObjeto();  
        output.writeObject(objProc);  
    }  
}
```

Após conectar, cria-se objetos para gerenciar o fluxo de dados entre cliente e servidor

Sockets TCP

Exemplo de aplicação

```
public class Servidor {  
    private ServerSocket servidor;  
    private Socket conexao;  
    private ObjectInputStream input;  
    private ObjectOutputStream output;  
    private Processa objProc;  
  
    public void ativaServer(){  
        servidor = new ServerSocket(56789, 10);  
  
        while(true)  
            conexao = servidor.accept();  
  
        output = new ObjectOutputStream(conexao.getOutputStream());  
        input = new ObjectInputStream(conexao.getInputStream());  
  
        objProc = (Processa) input.readObject();  
        objProc.processaObjeto();  
        output.writeObject(objProc);  
    }  
}
```

O servidor aguarda do cliente o objeto a ser processado, o processa e retorna o resultado

Sockets TCP

Exemplo de aplicação

```
public class Cliente {  
  
    private Socket client;  
    private ObjectInputStream input;  
    private ObjectOutputStream output;  
    private Imagem3D objImagem;  
  
    public void appCliente(){  
  
        client = new Socket(InetAddress.getLocalHost(), 56789);  
        output = new ObjectOutputStream(client.getOutputStream());  
        input = new ObjectInputStream(client.getInputStream());  
  
        objImagem = new Imagem3D();  
        output.writeObject(objImagem);  
        objImagem = (Imagem3D) input.readObject();  
  
    }  
}
```


Sockets TCP

Exemplo de aplicação

```
public class Cliente {  
  
    private Socket client;  
    private ObjectInputStream input;  
    private ObjectOutputStream output;  
    private Imagem3D objImagem;  
  
    public void appCliente(){  
  
        client = new Socket(InetAddress.getLocalHost(), 56789);  
        output = new ObjectOutputStream(client.getOutputStream());  
        input = new ObjectInputStream(client.getInputStream());  
  
        objImagem = new Imagem3D();  
        output.writeObject(objImagem);  
        objImagem = (Imagem3D) input.readObject();  
  
    }  
}
```

Envia uma mensagem de conexão ao servidor, e aguarda a aceitação

Sockets TCP

Exemplo de aplicação

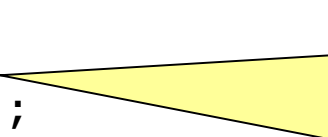
```
public class Cliente {  
  
    private Socket client;  
    private ObjectInputStream input;  
    private ObjectOutputStream output;  
    private Imagem3D objImagem;  
  
    public void appCliente(){  
  
        client = new Socket(InetAddress.getLocalHost(), 56789);  
        output = new ObjectOutputStream(client.getOutputStream());  
        input = new ObjectInputStream(client.getInputStream());  
  
        objImagem = new Imagem3D();  
        output.writeObject(objImagem);  
        objImagem = (Imagem3D) input.readObject();  
  
    }  
}
```

Analogamente ao servidor, cria-se os objetos para gerenciar o fluxo de dados

Sockets TCP

Exemplo de aplicação

```
public class Cliente {  
  
    private Socket client;  
    private ObjectInputStream input;  
    private ObjectOutputStream output;  
    private Imagem3D objImagem;  
  
    public void appCliente(){  
  
        client = new Socket(InetAddress.getLocalHost(), 56789);  
        output = new ObjectOutputStream(client.getOutputStream());  
        input = new ObjectInputStream(client.getInputStream());  
  
        objImagem = new Imagem3D();  
        output.writeObject(objImagem);  
        objImagem = (Imagem3D)input.readObject();  
  
    }  
}
```



Cria o objeto a ser processado, o envia ao servidor e aguarda o objeto de resposta