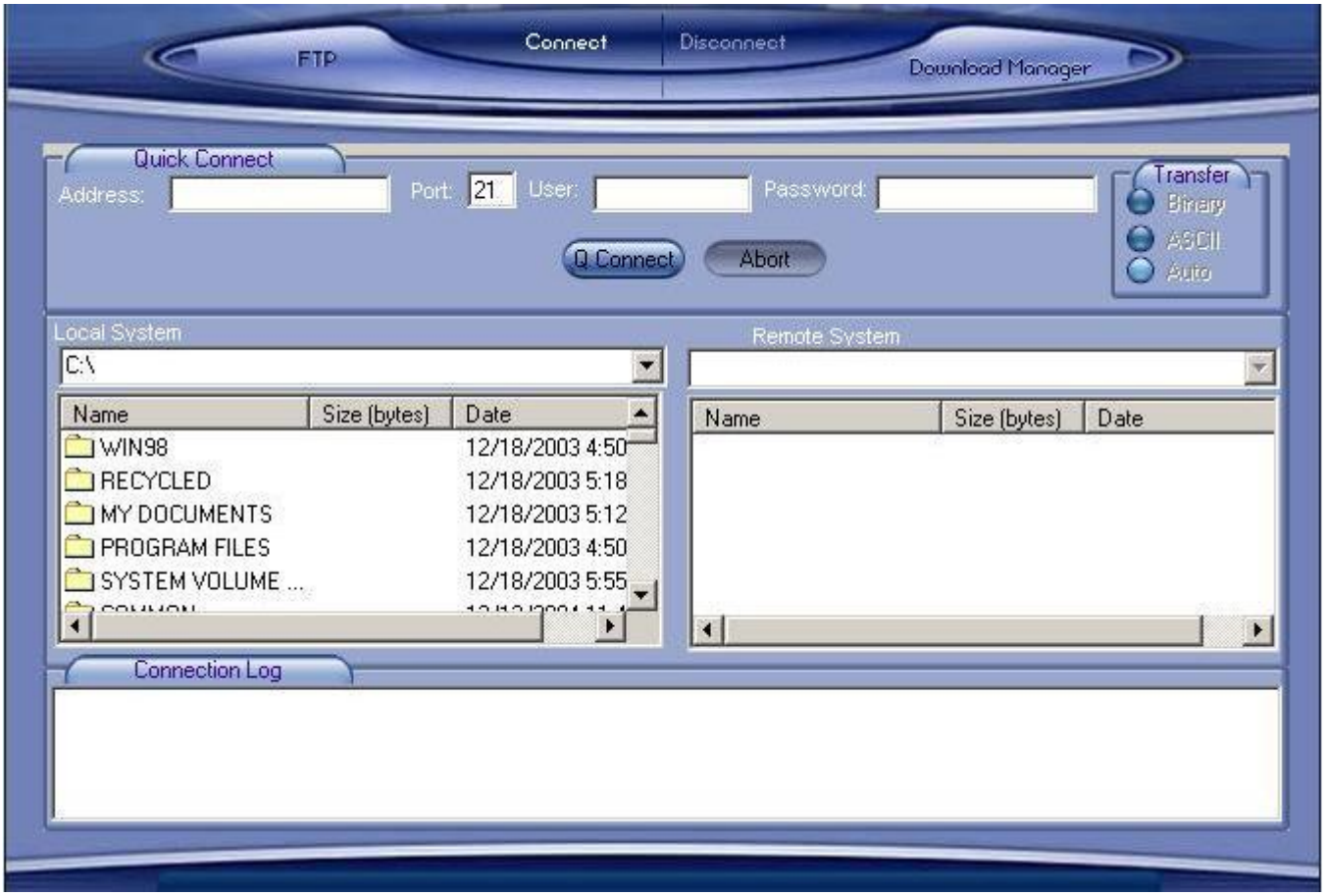
LAB 4 – Transferência de Arquivos

Descrição

Implemente nesta atividade de laboratório um programa em Java utilizando *threads* e *socket* TCP para localizar e baixar arquivos armazenados no disco rígido de uma **máquina remota**. O usuário através de uma interface (pode ser em linha de comando do DOS) deve ser capaz de listar os arquivos e diretórios da máquina remota e, caso queira, baixar o arquivo ou um diretório completo que quiser.

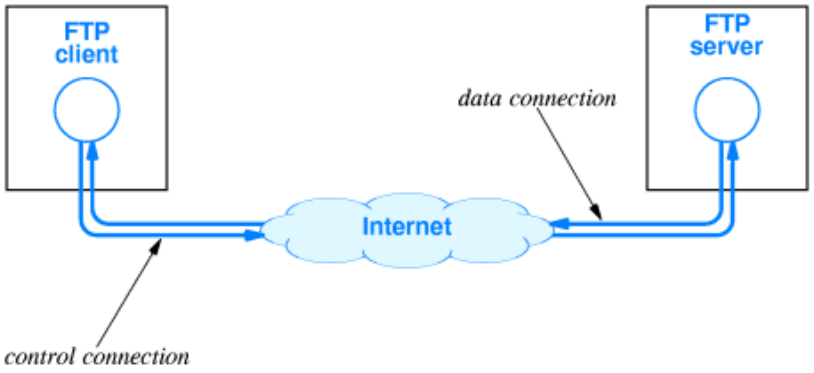


Implementação

O programa deve permitir que o usuário especifique critérios a serem considerados na busca de arquivos como nome, tipo (extensão), tamanho, data ou conteúdo (uma palavra ou frase contida no arquivo, por exemplo). Também deve ser especificado pelo usuário a partir de que ponto da árvore de diretórios será efetuada a busca (por exemplo, C:\, /usr/local, etc). A busca de arquivos deve ser efetuada no diretório especificado e também nos subdiretórios deste.

Crie um servidor TCP para receber conexões TCP clientes. Os clientes poderão se conectar a esse servidor especificando em linha de comando (ou numa interface mais elaborada com Java

Beans) o endereço IP e porta em que o servidor estiver atendendo. Uma vez feita a conexão, os clientes enviarão requisições (comandos) para fazer busca, navegar nos diretórios e baixar arquivos e diretórios.



A requisição (direção cliente para o servidor) deve estar encapsulada em um objeto Request, esse objeto deve conter como atributo a requisição e os parâmetros da requisição (p.ex. buscar arquivos com extensão .java ou baixar o arquivo filme.avi). O servidor ao receber o objeto Request deve extrair a requisição e executá-la. Ao termino da execução, o servidor deve criar a resposta (um objeto Reply) para enviar ao cliente. Dentro do objeto Reply deve conter um atributo que possibilite colocar um arquivo (p.ex. um array de bytes), caso a requisição seja de download. Use ObjectOutputStream e ObjectInputStream para serializar o objeto Request/Reply na comunicação cliente/servidor.

O servidor deve ser capaz de atender vários clientes ao mesmo tempo, ou seja, deve criar uma thread para cada cliente que se conectar a ele.

Para implementar a busca, utilizar o pacote java.io.File.

Pesquise as classes java.net.*, FileOutputStream, FileInputStream, ObjectOutputStream, ObjectInputStream para implementar o trabalho.

Apresentação

A atividade deve ser desenvolvida em duplas. O programa deve ser apresentado ao professor no laboratório até o dia 10/11. Os dois componentes do grupo devem estar presentes. Será verificado o funcionamento do programa e em seguida os alunos devem responder a questões sobre a forma como foram utilizados os sockets e o pacote java.io no programa.

Trabalhos entregues com atraso terão desconto automático de 2 pontos. Após a segunda semana (14 dias após o fim do prazo original), o trabalho não mais será aceito, ou seja, terá nota zero.

Podem ser atribuídas notas diferentes aos alunos de um grupo, dependendo das respostas às perguntas sobre o código do programa efetuadas pelo professor. Caso um dos alunos não esteja presente ou demonstre não conhecer o código do programa, será atribuída nota zero à atividade. Em caso de cópia do código de outro grupo, ambos terão nota igual a zero.

Java vê cada arquivo como um fluxo seqüencial de bytes.

Fluxos de arquivos podem ser utilizados para entrada e saída de dados como caracteres ou bytes.

ARQUIVOS CRIADOS BASEADOS NO FLUXO DE BYTES SÃO CHAMADOS ARQUIVOS BINÁRIOS.

ARQUIVOS CRIADOS BASEADOS NO FLUXO DE CARACTERES SÃO CHAMADOS DE ARQUIVOS TEXTO.

Java abre um arquivo criando e associando um objeto ao fluxo de bytes ou fluxo de caracteres.

Programas Java realiza o processamento de arquivos utilizando as classes no pacote java.io.

SERIALIZAÇÃO DE OBJETO

Às vezes, sabemos exatamente como os dados são armazenados em um arquivo. Nesses casos, podemos ler ou gravar um objeto inteiro a partir de um arquivo (lê do arquivo ou grava no arquivo). Como fazer isto ?

Java fornece a forma de fazer isso, o que é chamado de mecanismo de **serialização de objeto**.

Um **objeto serializado** é um objeto representado como uma **sequência de bytes** que inclui os dados do objeto, as informações de tipo do objeto e os tipos de dados armazenados no objeto.

Quando dizemos que um objeto é serializado, estamos afirmando que este objeto será transformado em bytes, e poderá ser armazenado em disco ou transmitido por um ***stream*** (fluxo de bytes).

O ***stream*** é um objeto de transmissão de dados, onde um fluxo de dados serial (fluxo de bytes) é feito através de uma origem e de um destino.

Assim, **um objeto serializado pode ser gravado num arquivo** e depois **lido do arquivo**, e **desserializado** para recriar o objeto na memória do computador, para sua execução.

Classes de fluxo são definidas no pacote java.io, como **FileInputStream** (para entrada baseada em bytes para um arquivo) e **FileOutputStream** (para saída baseada em bytes para um arquivo). Os arquivos são abertos criando objetos dessas classes de fluxo.

As classes **ObjectInputStream** e **ObjectOutputStream** permitem que objetos inteiros sejam lidos ou gravados em fluxo de bytes para um arquivo. Para realizar a serialização com arquivos, inicializamos os objetos **ObjectInputStream** e **ObjectOutputStream** com objetos de fluxo – **FileInputStream** e **FileOutputStream -** respectivamente.

Serialização de objetos serve para criarmos e manipularmos arquivos de acesso seqüencial. A serialização é realizada com fluxo baseado em bytes e assim arquivos seqüenciais criados e manipulados serão arquivos binários. Esses não podem ser visualizados nos editores de texto padrão e deste modo, temos que escrever um aplicativo separado para ler e exibir objetos serializados.

Vamos descrever dois tipos de ***stream***, o **FileOutputStream** e o **FileInputStream** para manipular objetos serializados.

O **FileOutputStream** é um fluxo de arquivo que permite a gravação em disco. Já o **FileInputStream** é justamente o contrário, permitindo a leitura de um arquivo em disco.

Para o nosso exemplo está descrita a **gravação / leitura de objetos serializados em um arquivo**.

Também serão utilizadas duas classes chamadas **ObjectInputStream** e **ObjectOutputStream**. Elas são responsáveis por **inserir e recuperar objetos serializados** do ***stream***.

Um objeto da classe **ObjectOutputStream** (com o método writeObject()) utiliza um objeto da classe **FileOutputStream** para gravar objetos serializados num arquivo:

Output = new **ObjectOutputStream**(new **FileOutputStream**("arquivo.ser"));

.ser é extensão de arquivo binário que contém objeto serializado.

FileOutputStream gera um arquivo para armazenar o objeto serializado.

ObjectOutputStream herda da interface **ObjectOutput** que tem o método **writeObject(),** e grava o objeto no arquivo.

Um objeto da classe **ObjectInputStream** utiliza um objeto da classe **FileInputStream** para ler objetos serializados num arquivo:

Input = new **ObjectInputStream**(new **FileInputStream**("arquivo.ser"));

FileInputStream carrega o arquivo

ObjectInputStream Lê o objeto serializado no arquivo.

ObjectInputStream herda da interface **ObjectInput** que tem o método **readObject(),** e lê o objeto serializado no arquivo.

Ver os exemplos Deitel, Ch14, exemplos 14.17 à 14.21 em downloads na página da disciplina.

Um outro exmplo:

Primeiro vamos desenvolver a classe chamada cliente, que será utilizada como objeto serial de armazenamento:

```
import java.io.Serializable;

// A classe deve implementar Serializable

public class Cliente implements Serializable

{
     private String nome;
    private char sexo;
    private String cpf;
     public Cliente( String nome, char sexo, String cpf )

    {

         super( );

         this.nome = nome;

         this.sexo = sexo;

         this.cpf = cpf;

    }



    public String getCpf( )

    {

         return cpf;

    }



    public void setCpf( String cpf )
```

```
{
    this.cpf = cpf;
}

public String getNome( )
{
    return nome;
}

public void setNome( String nome )
{
    this.nome = nome;
}

public char getSexo( )
{
    return sexo;
}

public void setSexo( char sexo )
{
    this.sexo = sexo;
}
public String toString( )
```

```
    {
        return this.nome + " / " + "Sexo: " + this.sexo + "\n" + "CPF:
"
            + this.cpf;
    }
}
```

Abaixo segue a main que executará a gravação e a leitura de objetos em um arquivo.

```
package br.com.artigos.serial;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class ExemploStream
{
    public static void main( String[] args )
    {
     // Cria o objeto serializado
     Cliente cliente = new Cliente("Glaucio Guerra",'M',"0000000001");
        try
        {
```

```
    //Gera o arquivo para armazenar o objeto serializado

    FileOutputStream arquivoGrav =

    new FileOutputStream("/Users/glaucio/Desktop/saida.dat");

    //  Generic data file created by a specific application; typically accessed only
    //    by the application that created the file; may contain data in text or binary
    //    format;  text-based DAT files can be viewed in a text editor.

    //Classe responsável por inserir os objetos

    ObjectOutputStream objGravar = new
        ObjectOutputStream(arquivoGrav);

    //Grava o objeto cliente no arquivo

    objGravar.writeObject(cliente);

    objGravar.flush();

    objGravar.close();

    arquivoGrav.flush();

    arquivoGrav.close();

    System.out.println("Objeto gravado com sucesso!");

}

catch( Exception e ){

        e.printStackTrace( );

}

System.out.println("Recuperando objeto: ");

try

{

    //Carrega o arquivo
```

```
            FileInputStream arquivoLeitura = new
                 FileInputStream(c:/saida.dat");

            //Classe responsável por recuperar os objetos do arquivo

            ObjectInputStream objLeitura =

                 new ObjectInputStream(arquivoLeitura);

            System.out.println(objLeitura.readObject());

            objLeitura.close();

            arquivoLeitura.close();

        }

        catch( Exception e ){

                e.printStackTrace( );

        }

    }

}
```

Saída:

```
Objeto gravado com sucesso!

Recuperando objeto:

Glaucio Guerra / Sexo: M

CPF: 0000000001
```

Existem diversas aplicações que utilizam o recurso de serialização. O Hibernate, streams para JME, e qualquer outro tipo de aplicação que trabalhe com fluxo de

I/O. Qualquer tipo de trabalho que envolver persistência ou tramitação de dados, a serialização é necessária.