Program #4

20 points; Due Nov. 20 (Wednesday) @ 5pm

Program Description

You will write the **Server** and the **Client** programs that run a simplified **File Transfer Protocol** for transferring files. The Server must be able to handle multiple clients simultaneously and listen on **port 5721** (control connection) for file transfer requests. Once a request is accepted, the Server opens **another port** (data connection) for the Client to establish the connection dedicated for file transfer (you can use 5720 or any port numbers between 5700-5800.) The data connection should be closed after the file transfer is done. However, the control connection must remain connected until the Client is disconnected. Each Client should be handled with a separate thread.

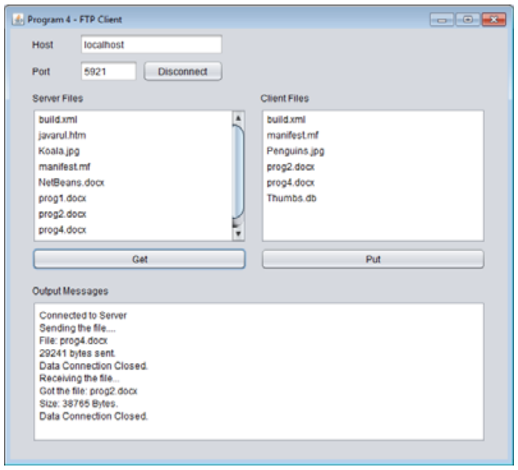
You have the OPTION to work on this program with a partner from any sections;

The Protocol

The simplified FTP supports only two commands: **GET [filename]** and **PUT [filename]**. A GET command transfers a file from the Server to the Client, and a PUT command transfers the file from the Client to the Server. The Client will initiate the connection and send either a GET or PUT command. The sequence of interactions between the Server and Client are summarized as follows.

1. **Client:** connect to the Server through the control connection 5721.
2. **Server:** accept the connection and send the list of files available for download to the Client.
3. **Client:** the UI displays the list of files from the Server. The Client sends either GET or PUT command as a request to download/upload the selected file, one at a time.
4. **Server:** receive the request, open a data connection dedicated to the file transfer, and send the port number to the Client. Close the data connection once the file transfer is done.
5. **Client:** use the specified port number to contact the Server for the file transfer, and close the data connection once the file transfer is done. Client UI must refresh and show the updated list of files. The Client can send another request through the control connection, or disconnect from the Server.

The following screen shots are sample Client UI with sample runs and the sample Server system output. You must use a **JList** to list the remote files in the folder **Files** from the Server, and a **JList** to list the local files from the Client's current folder.



Server Files:

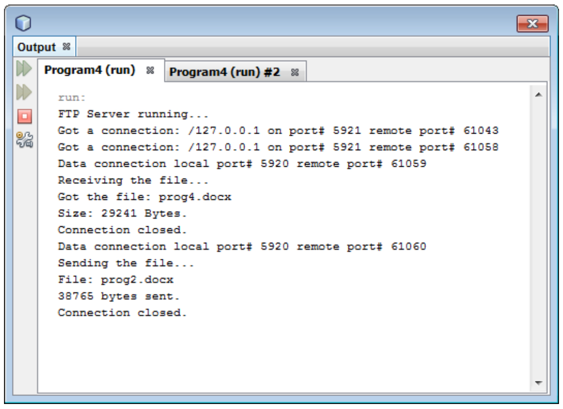
```
build.xml
javarul.htm
Koala.jpg
manifest.mf
NetBeans.docx
prog1.docx
prog2.docx
prog4.docx
```

Client Files:

```
build.xml
manifest.mf
Penguins.jpg
prog2.docx
prog4.docx
Thumbs.db
```

Output Messages:

```
Connected to Server
Sending the file....
File: prog4.docx
29241 bytes sent.
Data Connection Closed.
Receiving the file...
Got the file: prog2.docx
Size: 38765 Bytes.
Data Connection Closed.
```



```
run:
FTP Server running...
Got a connection: /127.0.0.1 on port# 5921 remote port# 61043
Got a connection: /127.0.0.1 on port# 5921 remote port# 61058
Data connection local port# 5920 remote port# 61059
Receiving the file...
Got the file: prog4.docx
Size: 29241 Bytes.
Connection closed.
Data connection local port# 5920 remote port# 61060
Sending the file...
File: prog2.docx
38765 bytes sent.
Connection closed.
```

Program Requirement

1. The Server's system output must show the Server is currently running, and display all file transfer activities. You must try-catch everything and display appropriate error messages identifying the specific exception. The Server must not crash under any situations.
2. You MUST follow the software development ground rules from CS2430.
3. You MUST demo your program by the demo due date, or **0 points**.
4. BOTH of you must attend the demo to get the credit for this program.
5. The Server must include 2 classes: **FTPServer** class and **FTPThread** class. You must use **JFrame** for the Client program. All files must be sent in **1024-byte chunks**. Follow the instructions below.

❖ Server Program

FTPServer class will be multi-threaded, where the processing of each incoming request will take place inside a separate thread of execution (**FTPThread**). You must have a **run()** method in FTPServer class.
FTPThread class implements a proprietary File Transfer Protocol that handles file transfers to/from the Clients. This class extends Java Thread class and you must implement the **run()** method that handles the file transfer. You must implement and use the following **private methods**.

- **listFiles()** – retrieve all file names (excluding folders) from the folder **Files** and send a list of file names with a single String instance to the Client socket. Here are some useful methods.

```
File dir = new File("Files");   // create a new instance of File class and set the pathname
File [] files = dir.listFiles();   // get the list of all files
files[i].isFile()  // returns true if files[i] is a file NOT a folder
files[i].getName()  //returns the file name as a String
```

- **sendFile(String** filename**)** – send the file **filename** thru the data connection. The file path will be **"Files\\" + filename**, if the folder **Files** contains the file. You must use double back-slash because a single back-slash identifies a control character next to it.
- **getFile(String** filename**)** – receive the file **filename** thru the data connection. The file path will be **"Files\\" + filename**, if the file is to be stored in the folder **Files**.

❖ Client Program

You must use **JFrame** and include the following components.

- **JTextField** for host IP/name and port number, and a **JButton** to connect/disconnect to/from the Server.
- **JList** for displaying the remote files on the Server and the local files on the Client.
- **JButton** for sending the GET or PUT command.
- **JTextArea** for displaying messages and the interactions between Client/Server.

When you bring up your Client, the local files must be automatically loaded to the JList. When the connect button is clicked, the remote files must be displayed on the JList. You must implement the following **private methods**.

- **listRemoteFiles()** – read the String sent from the Server, and use **StringTokenizer** to decompose the file names. Create an instance of Vector class and add all file names to the list. To export the names from the Vector to the JList, use **.setListData(Vector** v**)** method.
- **listLocalFiles()** – retrieve filenames from the local folder and list all files on the JList. Use an instance of Vector class to manage the list.
- **sendFile(String** filename**)** – send the file **filename** thru the data connection.
- **getFile(String** filename**)** – receive the file **filename** thru the data connection.

Program Grading

Exceptions/Violations	**Each Offense**	**Max Off**
Program not running	20	20
Failed to transfer files between Client and Server	20	20
**Either of the group members failed to show up for the demo (if work as a group of 2)	20	20
**Missing time logs (if work as a group of 2)	5	5
Failed to handle multiple connections	5	5
Improper list of files of Server/Client on UI	3	6
File transfer activities are not displayed properly	2	6
Exceptions not caught and handled	2	6
Not complying CS2430 programming ground rules	1	5

**Apply only to the people who worked as a group of 2.