VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Xiaoling Yu

IMPLEMENTATION OF A SECURE SHELL FILE TRANSFER PROGRAM IN JAVA

Information Technology
2010

FOREWORD

First of all, I would like to give big appreciation to my supervisor Dr. Ghodrat Moghadampour who has been patiently giving me valuable suggestions and big support whenever needed. In addition, I really appreciate my language teacher Dr. Ritva Rapila for guiding me with my communication skills from the moment I came to study in VAMK. I would also like to thank my boyfriend Olli Keskinen for personal support and technical advice in my thesis.

Moreover, I want to thank other teachers, especially Gao Chao, Antti Virtanen and Smail Menani for the quality education I have received. Finally I would like to thank all the people whose names are not mentioned here for their kindness which makes my stay in Vaasa so pleasant. I will definitely miss the precious time in the future.

I am honoured to graduate from VAMK as an Engineer.

Yu Xiaoling
19.02.2010

ABBREVIATIONS

API	Application Programming Interface
JVM	Java Virtual Machine
MVC	Model–View–Controller
SCP	Secure Copy
SFTP	SSH File Transfer Protocol
SSH	Secure Shell
FTP	File Transfer Protocol
IDE	Integrated Development Environment

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Programme of Information Technology

ABSTRACT

Author	Xiaoling Yu
Title	Implementation of a Secure Shell File Transfer Program in Java
Year	2010
Language	English
Pages	48
Supervisor	Ghodrat Moghadampour

With the popularity of secure shell (SSH) protocol, as well as the need of transferring files between a local computer and a remote computer, a simple and user friendly software is required to satisfy this need. Programs for SSH file transfer already exist, but they are often complex to use and are developed for single operating system.

Hence, the idea to develop such a software in Java resulted in this project of SSH File Transfer Program. By using SSH File Transfer Program user is able to transfer files from local computer to a server, and vice versa, with SSH protocol. Simple and user friendly interface allows user to do this easily. Java technology makes it possible to use this program on virtually any operating system that supports Java.

This application was developed with Java Swing technology and for the implementation of this project, Eclipse was chosen as the IDE (Integrated Development Environment). An open source SSH package, SSHTools (J2SSH), was used for the implementation of the SSH part of the program.

Keywords SSH, SFTP, SCP, File Transfer, Java, Java Swing

CONTENTS

1 INTRODUCTION

This project started as just an idea to have easy and secure way to transfer files between school and home. There are existing programs for transferring files through SSH protocol, such as WinSCP and SSH Secure Shell. Although they are secure as they use SSH protocol, they have functionalities that most people do not really use, and are complex to use because of this.

1.1 Rationale for This Project

The existing programs mentioned above are also made for a single operating systems, but a program made in Java can work on any operating system that supports Java. Even though the initial purpose for this project is to transfer files between school and home, there is no need to limit the usage only for that. This program is supposed to be a program that can be used for transferring files from and to any server that supports SSH file transfer.

The purpose of the project is to develop an SCP (Secure Copy Program) program that is used to transfer files from local computer to a server, and vice versa, with SSH protocol. Simple use friendly interface will allow user to transfer files easily between local and remote computer. Since the program is done in Java language, user is also able to manipulate files in other operating systems like Linux / MAC.

1.2 Technologies

This project uses number of technologies which are important to understand in order to be able to understand this thesis fully.

1.2.1 Java Language

Java is an object oriented programming language. Due to Java interpreters and runtime environments, known as Java Virtual Machines(VMs), compiled Java code can run on most different operating systems including Windows PCs, Macintoshes, and Unix computers.

There are many advantages why the software developers choose Java. Once Java software is written on one platform and it is possible for users to run it on virtually any other platform. This proves the idea “write once, run everywhere”. With the popularity of World Wide Web, Java language is well suited for the use on the World Wide Web since programs are created in Java can run within a Web browser and web services.

Besides, Java language can be used to develop server-side applications for online forums, stores, polls, HTML forms processing, and more. Using Java language to combine applications or services, it is possible to create highly customized applications or services. More important, Java language can offer the powerful and efficient applications for many digital devices, for example, most common used mobile phones, low-cost consumer products, and remote processors etc. [1]

1.2.2 Java Swing

Swing is used to build a Java program with a graphical user interface (GUI). Swing is part of Java Foundation Class (JFC). Swing is not a replacement for Abstract Window Toolkit (AWT), actually it is built on top of the core AWT libraries to provide a more sophisticated set of GUI components. [2]

Swing has two key features: Lightweight Components and Pluggable Look and Feel. Lightweight component feature makes it more efficient use of resources since a lightweight component does not depend on any native system classes (commonly called as “peer” classes). Java Look- and-Feel classes support most of components to have their own view in Swing.

Swing's pluggable Look- and-Feel feature makes it so that the user does not need to restart the application in order to switch the Look- and-Feel of Swing components. For Swing library supports a cross-platform look and feel so that no matter where the program runs, it keeps the same across all platforms. [3]

1.2.3 SSH

SSH (Secure Shell) is a network protocol used to create a secure connection. SSH is mostly used on Linux and Unix based systems. However, SSH is also available for Windows, Macintosh, and OS/2. SSH uses public-key to authenticate the remote computer and allow the remote computer to authenticate the user, if necessary. Hence, SSH has strong authentication and secure communications over insecure channels. Besides, SSH provides encryption so that it is almost impossible for an outsider to steal passwords when using SSH to log in session.

SSH protocol can be used for many applications. Here are some examples that show the uses of SSH, to login to a shell on a remote host , to execute a single command on a remote host, to copy files from a local server to a remote host, etc. [4]

1.2.4 SSH File Transfer Protocol (SFTP)

SSH File Transfer Protocol, sometimes called Secure File Transfer Protocol or SFTP, is a secure protocol and it uses an encrypted network connection provided by Secure Shell (SSH). SFTP is a replacement for the original SCP(Secure Copy) and it was designed to provide file transfer and manipulation functionality over any reliable data stream. [5]

SFTP is functionally prior to FTP, also, it is more platform independent than FTP. However, compared with File Transfer Protocol (FTP), SFTP has lower speed since it has encryption and necessity to wait for packet confirmations. [6]

1.2.5 SSHTools (J2SSH)

SSHTools (J2SSH) is a suite of Java SSH applications providing a Java SSH API, SSH Terminal, SSH secured VNC client, SFTP client and SSH Daemon. It offers a cross platform toolkit for all areas of SSH development. With J2SSH it is possible to create both client and server applications as well as develop extensions for J2SSH enabled applications. [7]

2 APPLICATION DESCRIPTION

The main functionalities of this program of course are the download and upload, but other functionalities are critical for this kind of program to be usable as well.

2.1 Requirement Analysis

Here is a list of functions which the program must include:

- Browse the local and remote folders
- Download files: download files from a server to local computer
- Upload files: upload files to a server from local computer

The program should have the following functions:

- Cancel transferring files: cancel the transferring action when a file is being transferred
- Delete files: delete local and remote files and folders
- Rename file: rename local and remote file and folder
- Create new directory: Create new directory in local computer or remote server

Nice to have functions:

- Change permissions: change the permissions to specify who and what can be read, write, modify and access different files and directories in remote file system.
- Drag & Drop functionality

2.2 Accessing the functions

The functions should be accessed easily in the normal way in which functions are usually accessed in programs. Changing to a folder should work by double clicking the folder, or by using arrow keys to go over the folder and pressing enter key. Going to the parent folder should work with a tool bar button and also with backspace key. There should be a tool bar button for a quick way to change to default home folder.

Creating new folder should work through menu and through a button on the tool bar. Renaming should work through menu and through the F2 key which normally is used for renaming. Deleting should work through menu and through delete key. Uploading and downloading should be accessed through menu and there should also be tool bar buttons for them.

In addition to the methods mentioned above, there should also be a popup menu which appears when right mouse button is clicked. This menu should include functions like create new folder, rename, delete, upload, download and edit permissions.

2.3 User Interface

There should be a login window that user can input server address, server port, user name and password If the given information are not valid, an error message should be shown telling what information is invalid. If all of the information is valid, the connection should be established and main window opened.

The main window should be divided in two parts. On the left side, there should be the list of local files. On the right side, there should be the list of remote files. There should be tool bar above both of the file lists, to access some of the above mentioned functions. The main window should also have menu bar from where all functions can be accessed.

If rename or new folder function is started, there should be a small window which has a text field where user can type the new name of the file or folder. If delete is pressed, there should be a confirmation where user is asked whether he/she is sure that he/she wants to delete the selected files. If user presses yes, the files are deleted. If user presses no, the program should continue in the normal way without any operation.

If user selects download or upload, a window should appear showing the progress of the download or upload. The name of the current file or folder that is being uploaded or downloaded should be shown. There should be also information of how many files or folders are still waiting to be downloaded or uploaded. There should be a cancel button from where the user can cancel the download or upload, and this should stop the process immediately. The files that are already downloaded or uploaded so far should remain and not be deleted when cancel is pressed.

If user wants to look at file properties, a new window should again appear showing the detailed information about the file. If the file is in remote file system, this window should also include an option to change the permissions of the file.

3 ANALYSIS AND DESIGN

For software applications, the first and the most important thing is to analyse and design the program before implementation. It is important to have well-designed system since it benefits any program and makes the implementation easier.

3.1 UML Diagrams

The UML design of this program is done with Borland Together.

3.1.1 Main Functions

From the following figure, which is a use case diagram, we can see the main functions of the application.

We can see there is only one actor which is user self. User is able to login to the program, upload files, download files, cancel the transfer, delete files, rename file, change file permissions as well as create new directory. All of these cases also apply for folders.

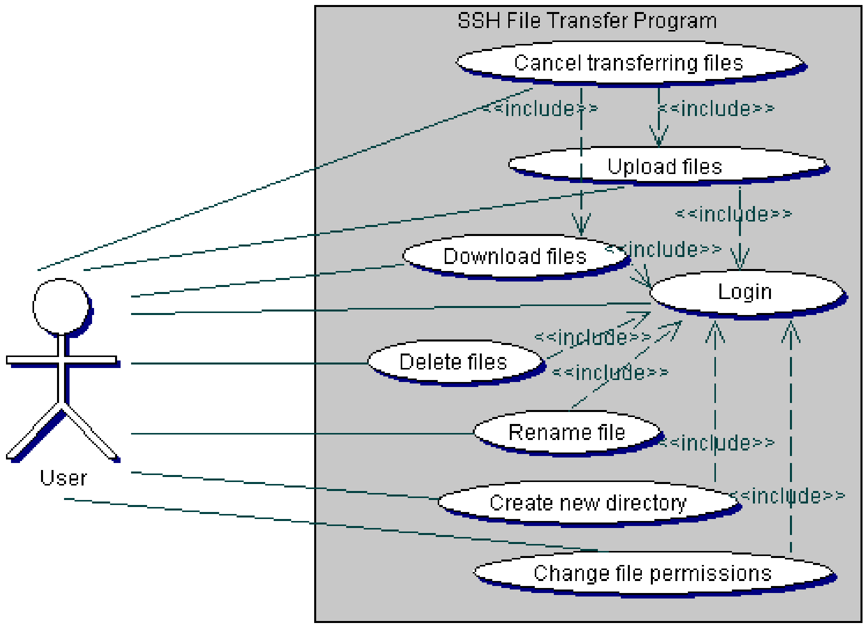


Figure 1: Use Case Diagram

3.1.2 Application Main Modules

Application is divided into three packages: **ui** , **util** and **filemanipulation**. Ui package is for all graphical user interface files and the filemanipulation package is for classes that work between UI and files, especially manipulate them and get the file lists etc. The package util has all the remaining classes, that are used in support of ui and filemanipulation.

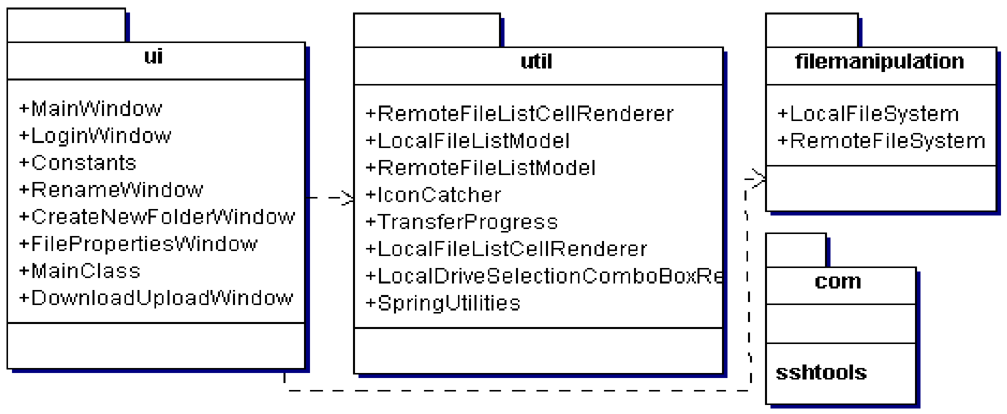


Figure 2: Component diagram

3.1.3 Class Relationships

In UI package, all window classes, except the DownloadUploadWindow, is called by MainWindow. Constants is used by MainWindow and LoginWindow. DownloadUploadWindow is called by RemoteFileSystem when download or upload is started. In FileManipulation package, there are no class relationships besides the already mentioned DownloadUploadWindow.

In Util package, all renderers and models are used by MainWindow of UI package. IconCatcher is used by all renderers to get file icons. SpringUtilities is used by LoginWindow to create the layout. TransferProgress is used by DownloadUploadWindow, to show the progress of the transfer.

3.1.4 Class Diagrams

Here we take a more detailed look at the classes and roles of each class.

Figure 3 shows the class diagram in package filemanipulation. There are two classes: **RemoteFileSystem** and **LocalFileSystem**. One method contained in both of these classes is listFiles. This is the method which retrieves the list of files in the current working directory so that they can be shown on the screen. Other methods that both of these classes contain are the methods used for file manipulation, such as deleteFiles, renameFile, createNewFolder.

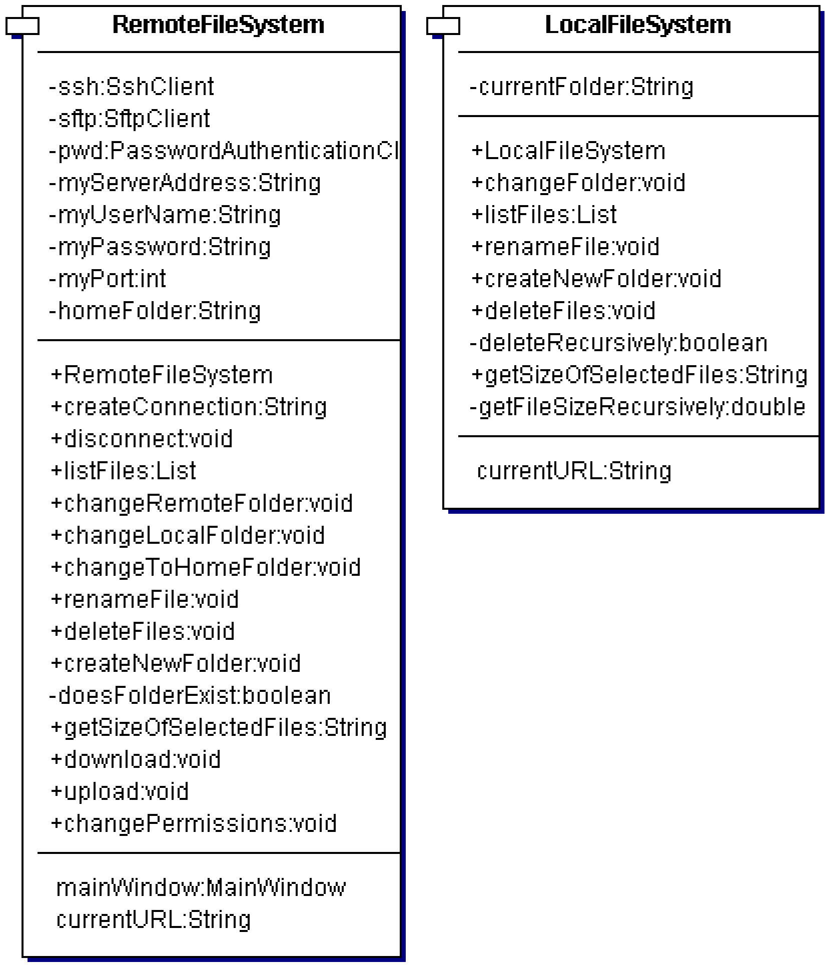


Figure 3: Class diagram of filemanipulation package

In addition to these methods, RemoteFileSystem also has important role in creating the connection to the remote server. This is done using the

createConnection method. After connection is created, the object of SftpClient is stored in the instance of RemoteFileSystem and used in all other operations that involve the remote system.

Shown in Figure 4 is the class diagram in package ui. There are 8 classes in this package. Most of these classes represent a window in the graphical user interface. The class called **MainClass** starts the program and it will launch **LoginWindow**. Class LoginWindow takes care of the valid connection. **MainWindow** is the window which shows the local and remote files, and includes menu, tool bar and status bar.

MainWindow consists of mainly listener methods which are launched when user interacts with the graphical user interface. These listener methods, such as actionPerformed and keyPressed then call another method, such as download, which in turn calls the appropriate method from the RemoteFileSystem. The reason why methods such as download are not directly called from RemoteFileSystem from actionPerformed is to avoid having to code same part multiple times. Many actions can be accessed through many places (clicking from menu, using keyboard), so it is sensible to create one method for one action, which is then called from the listener.

RenameWindow is a very simple window having only one text field and two buttons (OK and Cancel). **CreateNewFolderWindow** has same outlook as RenameWindow but is used for the user to give the name of new folder instead of renaming an existing file or folder. **FilePropertiesWindow** creates a window that shows the properties of a file or folder. It has labels for file name, size, etc. For remote file, it also has checkboxes for permissions.

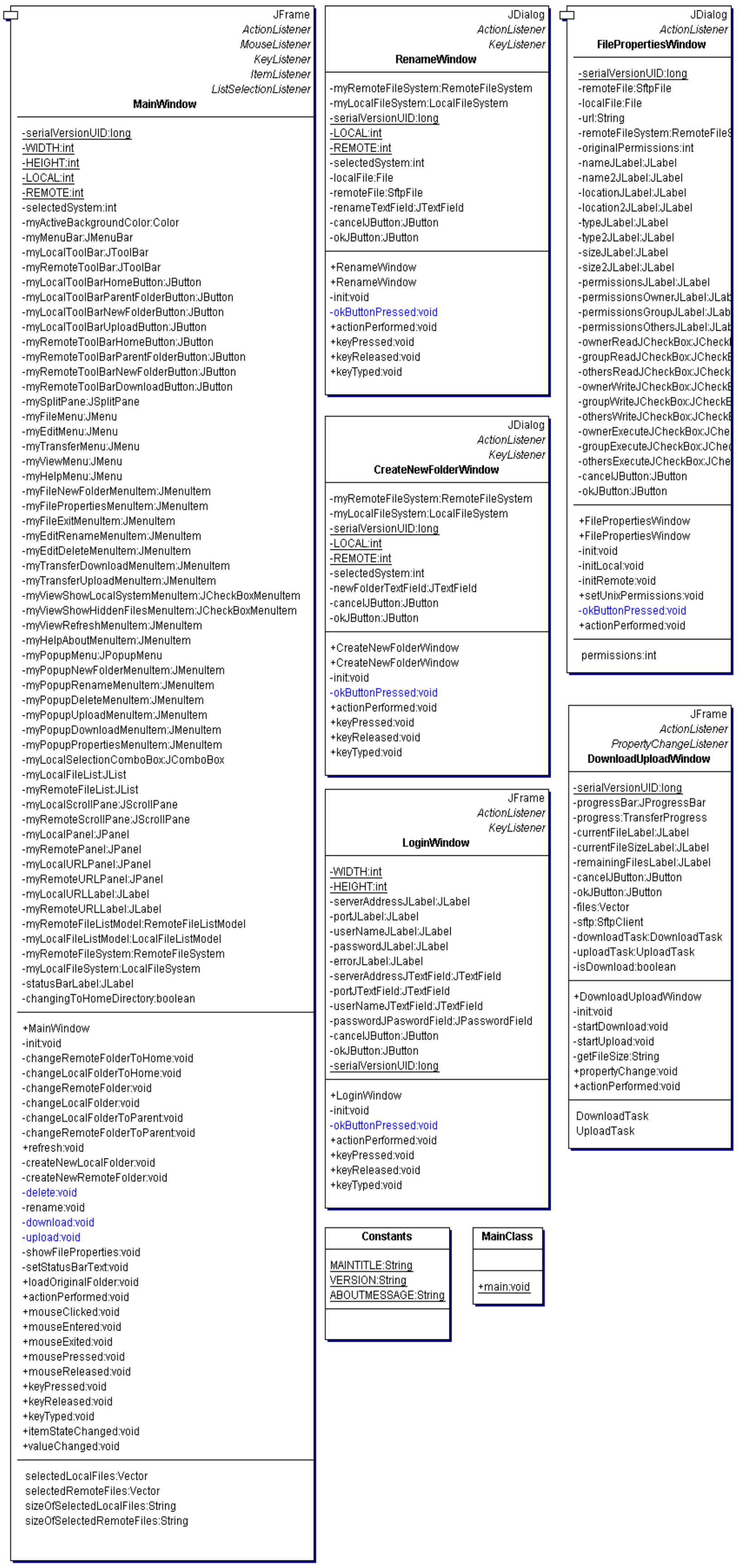


Figure 4: Class diagram of ui package

DownloadUploadWindow is a window that shows the progress of an ongoing download or upload. In addition to having a progress bar that shows the percentage of current upload, it also shows the name of the file that is being downloaded or uploaded, and number of remaining files or folders. DownloadUploadWindow also contains two inner classes: **DownloadTask** and **UploadTask**. These classes extend SwingWorker, which allows the download and upload progress to operate in a separate thread. This allows the GUI to operate, and user to press cancel button, while the progress is going on in the background.

Constants is a class which holds some constant text values that are used by some windows. It for example holds the name and version of the program, and is used by LoginWindow and MainWindow to display this information. These texts are placed in this Constants class so that changing of some of the texts can be made easily just in one class instead of having to modify same text in each class one by one.

Util package has also 8 classes. Class **RemoteFileListModel** extends AbstractListModel and acts as a model for JList component. The responsibilty of a this model is to store the list of objects, in this case SftpFile objects (SftpFile is a class in J2SSH), for the JList component that shows the list of remote files. Class **RemoteFileListRenderer** is responsible to render the components of the JList to the screen. Without this renderer, JList would not show the files correctly (their names), and there would not be icons for each file.

LocalFileListModel and **LocalFileListRenderer** are doing the same thing for the JList component of the local file list. Instead of having the objects of SftpFile class, the LocalFileListModel stores instances of java.ui.File class. Both of the renderers, RemoteFileListRenderer and LocalFileListRenderer use the **IconCatcher** to get the icon from the user's local icons. Each file type has their own icon that represents the type of file (or folder). IconCatcher is also used by the **LocalDriveSelectionComboBoxRenderer**, which is responsible for rendering the elements inside the JComboBox which is used for selecting the local drive.

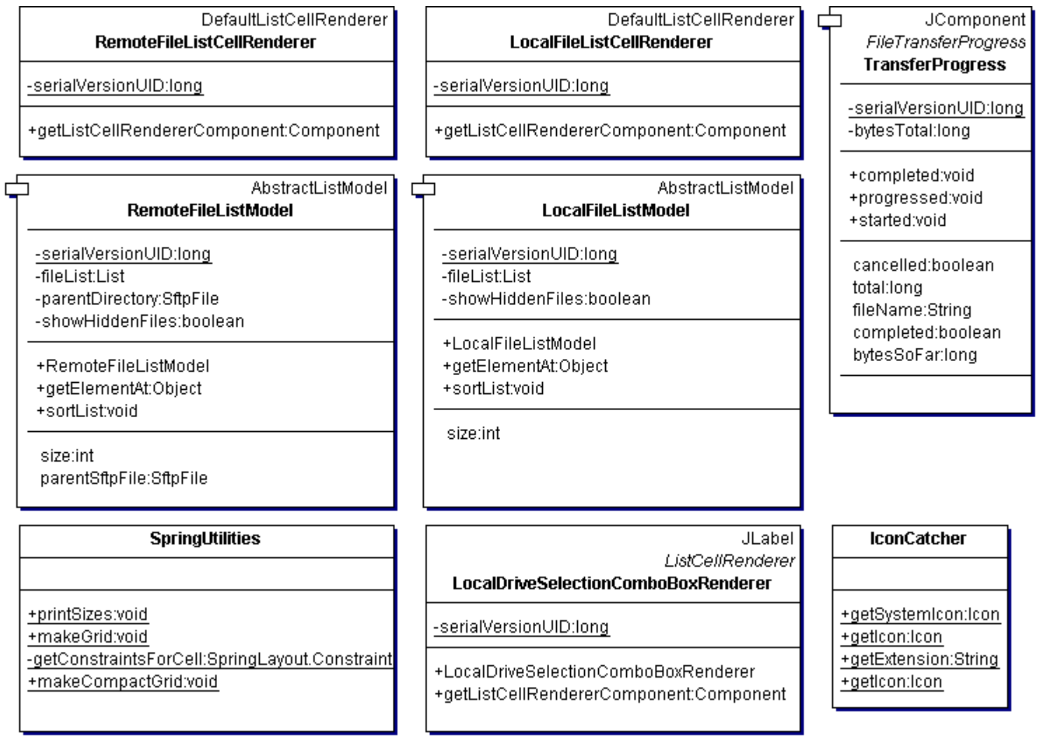


Figure 5: Class diagram of util package

TransferProgress class is an implementation of FileTransferProgress interface, provided by J2SSH package. This class is used to get the information about the progress of an upload or download. An instance of this class is given as a parameter to the methods in one of the J2SSH classes for uploading and downloading. That then automatically uses the methods in TransferProgress to update the information about the progress, and can be then used by the graphical user interface to show the progress.

SpringUtilities is a class made by Sun Microsystem, the producer of Java, and is used by SpringLayout which is one of the default layouts in Java. Sometimes Sun Microsystem provides implementation of some classes that are not provided readily in Java's default classes. These classes complement the default Java classes. This class has not been modified by me at all. [8]

To get a better idea of the renderers and models, it can be understood as model-view-controller architecture. In model-view-controller architecture, the system is divided into three parts. Model represents the data, view represents the presentation (graphical user interface), and controller is the part which controls everything Controller decides what is shown to the user through view, and controller receives the input from user and communicates with controller on necessary changes. Figure 6 illustrates the architecture of model-view-controller.

In Java, JList component uses a design that represents model-view-controller architecture. JList itself is the controller, while it has a separate model-object which stores the actual data, and renderer object which is used to draw the data to graphical user interface. In my case, the model and renderer have been replaced by non-default classes. LocalFileListModel and RemoteFileListModel are the models, which are holding the data, and LocalFileListRenderer and RemoteFileListRenderer represent the view, which are responsible for rendering the data.

All of these classes extend a default Java classes and overwrite a method: getListCellRenderer-method for renderers, and getElementAt for models. JList uses these classes by calling the overwritten methods.

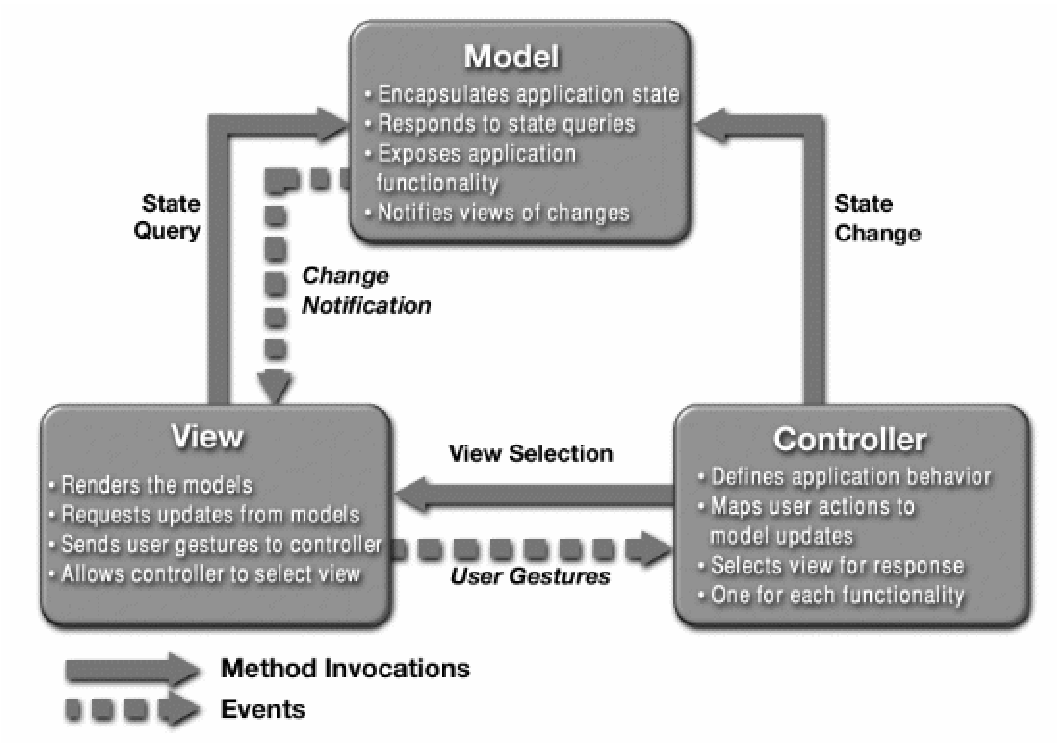

Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Figure 6: Model-View-Controller architectural pattern

[9]

3.1.5 Detailed Descriptions of Main Functions

Here are are the sequence diagrams for main operations in my application From Figure 7-Login Sequence Diagram, we can see how login method is implemented.

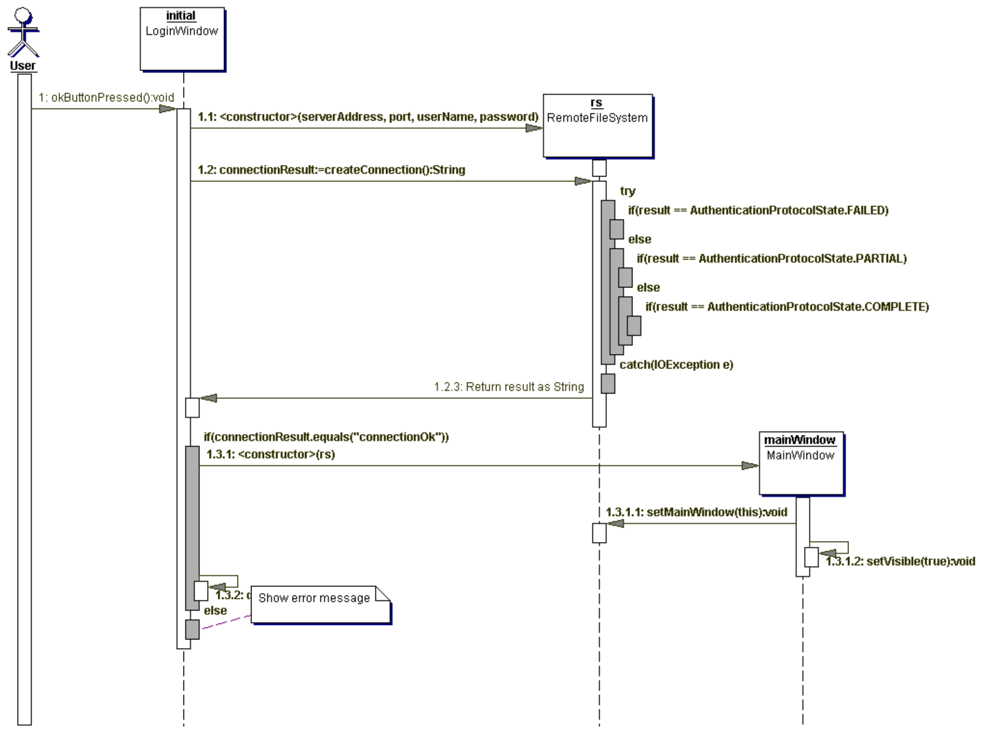

User
initial
LoginWindow
1: okButtonPressed():void
1.1: <constructor>(serverAddress, port, userName, password)
rs
RemoteFileSystem
1.2: connectionResult:=createConnection():String
try
if(result == AuthenticationProtocolState.FAILED)
else
if(result == AuthenticationProtocolState.PARTIAL)
else
if(result == AuthenticationProtocolState.COMPLETE)
catch(IOException e)
1.2.3: Return result as String
if(connectionResult.equals("connectionOk"))
1.3.1: <constructor>(rs)
mainWindow
MainWindow
1.3.1.1: setMainWindow(this):void
1.3.1.2: setVisible(true):void
1.3.2:
else
Show error message

Figure 7: Sequence diagram of login function

MainClass starts the LoginWindow constructor. When OkButton is pressed(action is performed) in LoginWindow, it will launch RemoteFileSystem constructor. The information of server address, port, username and password are taken from the LoginWindow's textfields and passed on to the instance of RemoteFileSystem in the constructor call.

After this, createConnection-method is called with the instance of RemoteFileSystem that was just created. It tries to establish the connection, using a method in a class of J2SSH. This method returns the result of the attempt of connection and this result is used to create an error string, or in case of successful connection, a "connectionOk" string is passed to the string. This string is then returned to the LoginWindow.

If the returned string contains “connectionOk”, LoginWindow creates an instance of MainWindow and hides itself. If returned string was something else, it means it contains the error message and it is shown on a label in LoginWindow.

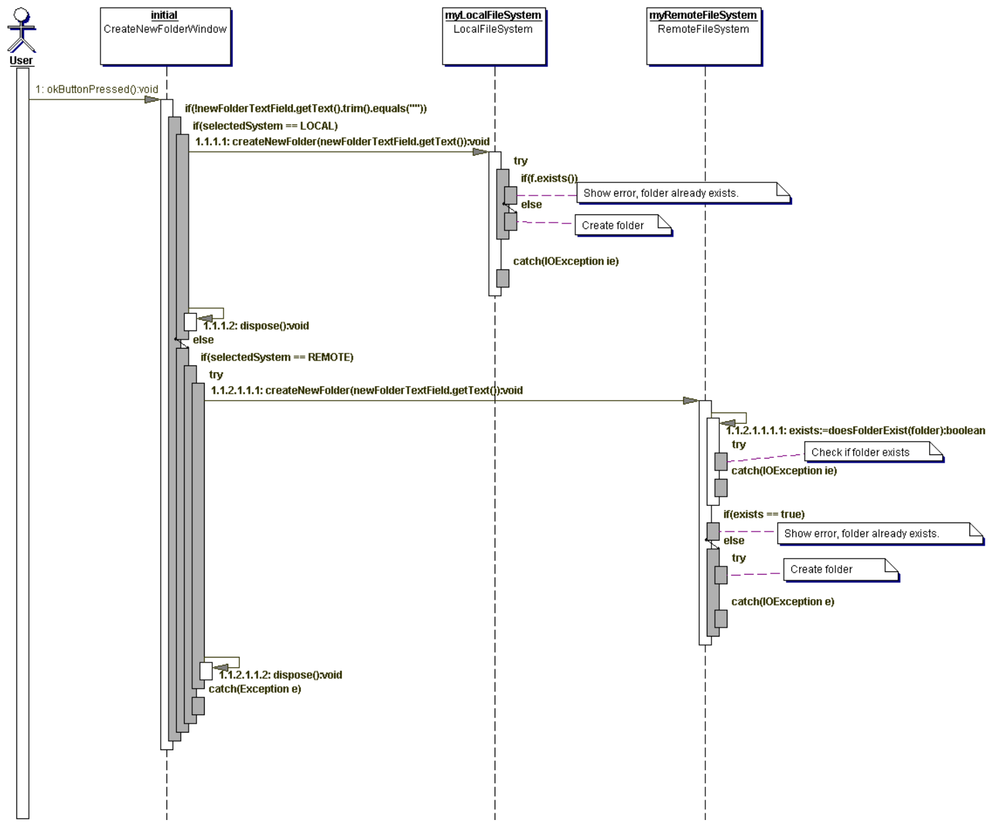

User
initial
CreateNewFolderWindow
myLocalFileSystem
LocalFileSystem
myRemoteFileSystem
RemoteFileSystem
1: okButtonPressed():void
if(!newFolderTextField.getText().trim().equals(""))
if(selectedSystem == LOCAL)
1.1.1.1: createNewFolder(newFolderTextField.getText()):void
try
if(f.exists())
Show error, folder already exists.
else
Create folder
catch(IOException ie)
1.1.1.2: dispose():void
else
if(selectedSystem == REMOTE)
try
1.1.2.1.1.1: createNewFolder(newFolderTextField.getText()):void
1.1.2.1.1.1.1: exists:=doesFolderExist(folder):boolean
try
Check if folder exists
catch(IOException ie)
if(exists == true)
Show error, folder already exists.
else
try
Create folder
catch(IOException e)
1.1.2.1.1.2: dispose():void
catch(Exception e)

Figure 8: Sequence diagram of create new folder function

Figure 8 shows how “create new folder” function is implemented. MainWindow starts the CreateNewFolderWindow constructor. When OkButton is pressed (action is performed) in CreateNewFolderWindow, it will first check that the new folder's name is not empty, and then it checks if the selected system is local or remote.

If selected system is local, createNewFolder method from myLocalFileSystem (instance of LocalFileSystem) will be called, and then it checks if the folder with same name exists or not. If such folder already exists, error message will be shown, otherwise a new folder will be created. If the selected system is remote, it works in the same way as local system. createNewFolder method from myRemoteFileSystem (instance of RemoteFileSystem) will be called, and then it checks if the folder with same name exists or not. If yes, error message will be shown, otherwise a new folder will be created.

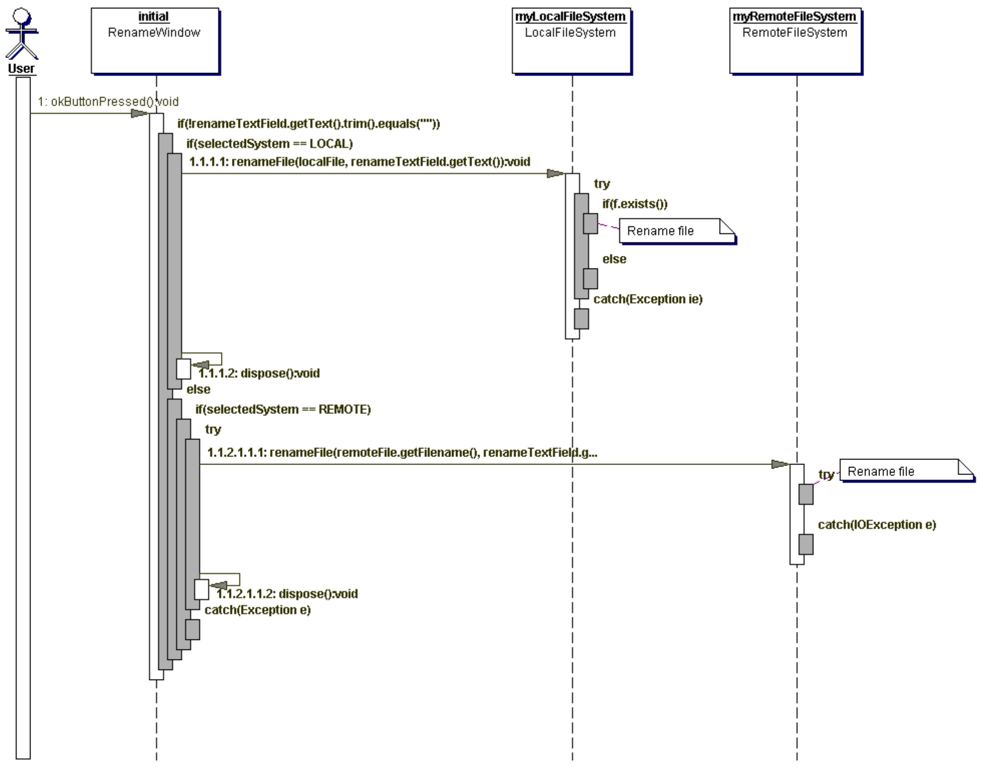

User
initial
RenameWindow
myLocalFileSystem
LocalFileSystem
myRemoteFileSystem
RemoteFileSystem
1: okButtonPressed():void
if(!renameTextField.getText().trim().equals(""))
if(selectedSystem == LOCAL)
1.1.1.1: renameFile(localFile, renameTextField.getText()):void
try
if(f.exists())
Rename file
else
catch(Exception ie)
1.1.1.2: dispose():void
else
if(selectedSystem == REMOTE)
try
1.1.2.1.1.1: renameFile(remoteFile.getFilename(), renameTextField.g...
try
Rename file
catch(IOException e)
1.1.2.1.1.2: dispose():void
catch(Exception e)

Figure 9: Sequence diagram of rename function

Figure 9 shows how “Rename” function is implemented. MainWindow starts the RenameWindow constructor. When OkButton is pressed (action is performed) in RenameWindow, it will first check the inputted new name is not empty, and then it checks if the selected system is local or remote.

If selected system is local, renameFile method from myLocalFileSystem will be called, and then it checks if the file/folder with same name exists or not. If there is problem with name overlapping, error message will be shown, otherwise a file's/folder's name will be changed.

If the selected system is remote, it works in the same way as local system. renameFile method from myRemoteFileSystem will be called, and then it checks if the file/folder with same name exists or not. If there is problem with name overlapping, error message will be shown, else a file's/folder's name will be changed.

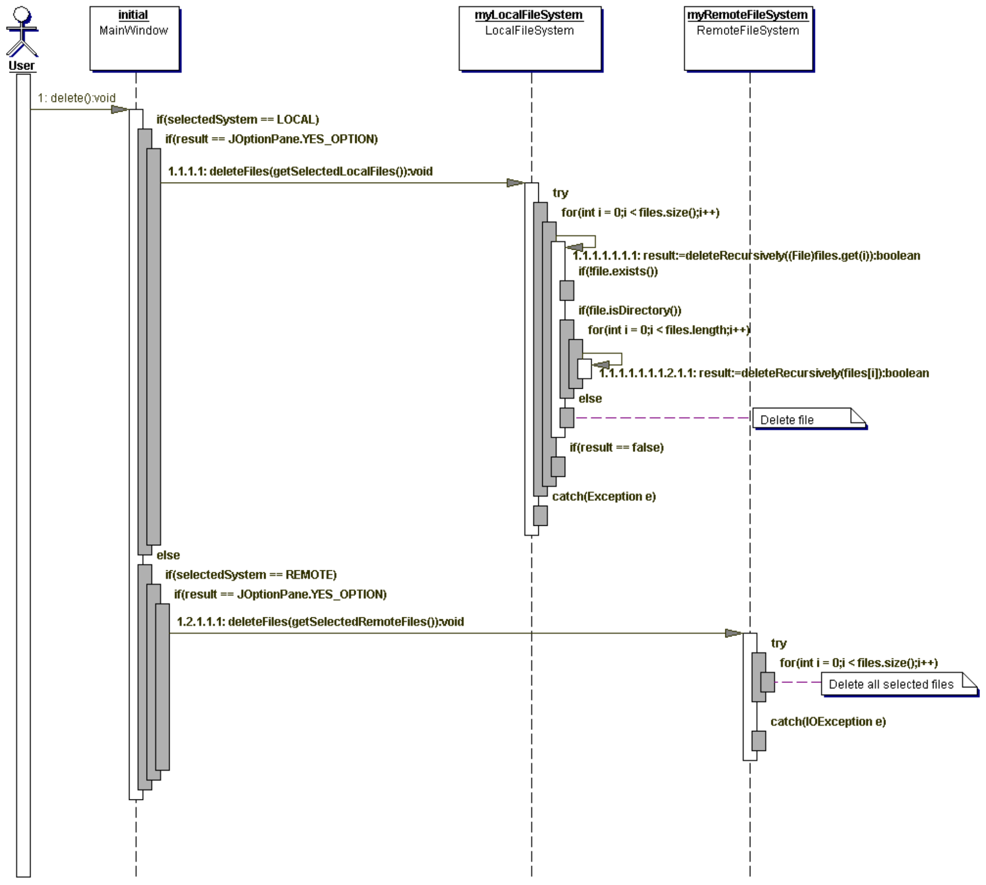

User
initial
MainWindow
myLocalFileSystem
LocalFileSystem
myRemoteFileSystem
RemoteFileSystem
1: delete():void
if(selectedSystem == LOCAL)
if(result == JOptionPane.YES_OPTION)
1.1.1.1: deleteFiles(getSelectedLocalFiles()):void
try
for(int i = 0;i < files.size();i++)
1.1.1.1.1.1.1: result:=deleteRecursively((File)files.get(i)):boolean
if(!file.exists())
if(file.isDirectory())
for(int i = 0;i < files.length;i++)
1.1.1.1.1.1.1.2.1.1: result:=deleteRecursively(files[i]):boolean
else
Delete file
if(result == false)
catch(Exception e)
else
if(selectedSystem == REMOTE)
if(result == JOptionPane.YES_OPTION)
1.2.1.1.1: deleteFiles(getSelectedRemoteFiles()):void
try
for(int i = 0;i < files.size();i++)
Delete all selected files
catch(IOException e)

Figure 10: Sequence diagram of delete function

Figure 10 shows how “Delete” function is implemented. When delete method is called in MainWindow, it checks if the selected system is local or remote. Next a dialog made with JOptionPane with delete confirmation is shown to user. If selected system is local, and user selected yes option, deleteFiles method with parameters “getSelectedLocalFiles” (a method that returns the files/folders that are selected in the file list as array of File objects) will be called from myLocalFileSystem. Inside delete method all files/folders are for-looped through and each is passed to a method which deletes them recursively.

If selected system is remote, and user selects yes from the dialog, deleteFiles method with parameters “getSelectedRemoteFiles” will be called from myRemoteFileSystem. J2SSH has ready method for deleting folders recursively, so special method for recursive deletion is not needed.

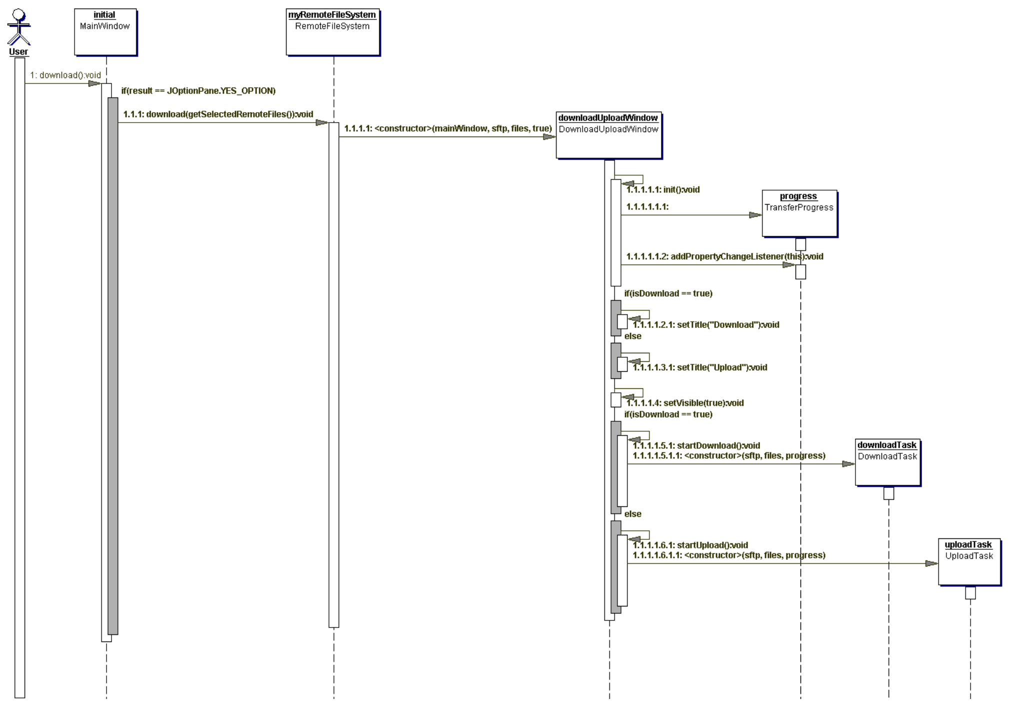

User
initial
MainWindow
myRemoteFileSystem
RemoteFileSystem
1: download():void
if(result == JOptionPane.YES_OPTION)
1.1.1: download(getSelectedRemoteFiles()):void
1.1.1.1: <constructor>(mainWindow, sftp, files, true)
downloadUploadWindow
DownloadUploadWindow
1.1.1.1.1: init():void
1.1.1.1.1.1:
progress
TransferProgress
1.1.1.1.1.2: addPropertyChangeListener(this):void
if(isDownload == true)
1.1.1.1.2.1: setTitle("Download"):void
else
1.1.1.1.3.1: setTitle("Upload"):void
1.1.1.1.4: setVisible(true):void
if(isDownload == true)
1.1.1.1.5.1: startDownload():void
1.1.1.1.5.1.1: <constructor>(sftp, files, progress)
downloadTask
DownloadTask
else
1.1.1.1.6.1: startUpload():void
1.1.1.1.6.1.1: <constructor>(sftp, files, progress)
uploadTask
UploadTask

Figure 11: Sequence diagram of download function

Figure 11 shows how “Download” method is implemented. When download method is called in MainWindow, a confirmation window to ask if user is sure is shown. If user selects yes, download method from RemoteFileSystem is called. Selected files (array of SftpFiles) are given as parameter. RemoteFileSystem then starts the DownloadUploadWindow. First the window is initiated, and instance of TransferProgress is created.

Next one instance of DownloadTask, which is an inner class of DownloadUploadWindow, is started. DownloadTask is a class which keeps the download process in a separate thread, so that window can still be updated and cancel button can be pressed while download is going on. Inside DownloadTask, a method from J2SSH is used for downloading. This method is given the instance of

TransferProgress as parameter, so it then updates the attributes inside the TransferProgress. Every time attribute has changed in the TransferProgress, PropertyChangedListener launches method in the DownloadUploadWindow. Progress bar and other information about the download are then updated in GUI.

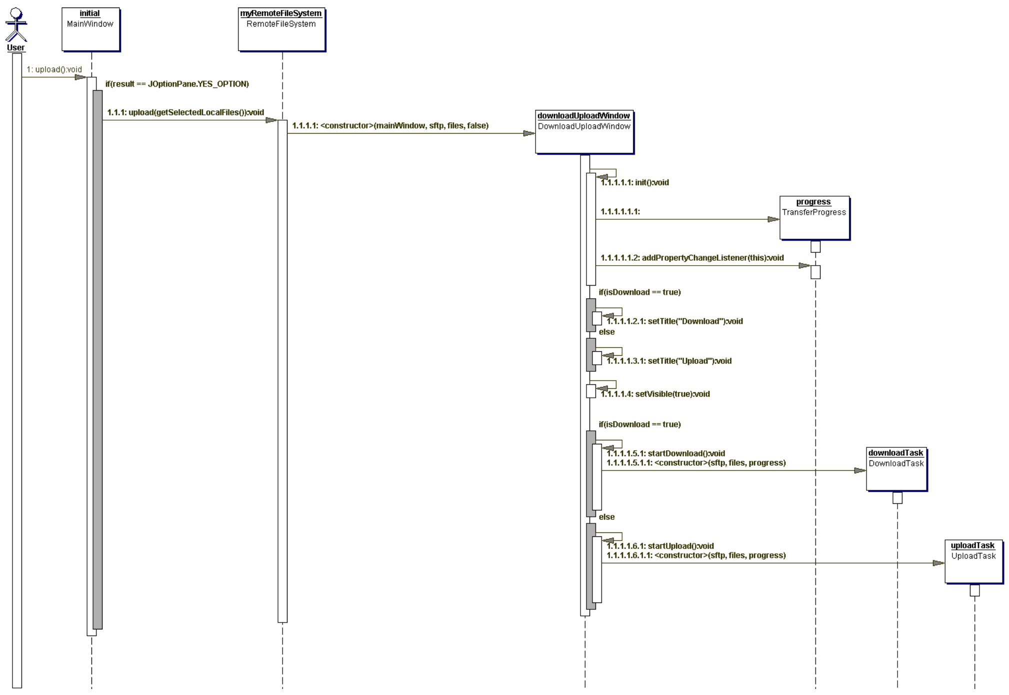

User
initial
MainWindow
myRemoteFileSystem
RemoteFileSystem
downloadUploadWindow
DownloadUploadWindow
progress
TransferProgress
downloadTask
DownloadTask
uploadTask
UploadTask
1: upload():void
if(result == JOptionPane.YES_OPTION)
1.1.1: upload(getSelectedLocalFiles()):void
1.1.1.1: <constructor>(mainWindow, sftp, files, false)
1.1.1.1.1: init():void
1.1.1.1.1.1:
1.1.1.1.1.2: addPropertyChangeListener(this):void
if(isDownload == true)
1.1.1.1.2.1: setTitle("Download"):void
else
1.1.1.1.3.1: setTitle("Upload"):void
1.1.1.1.4: setVisible(true):void
if(isDownload == true)
1.1.1.1.5.1: startDownload():void
1.1.1.1.5.1.1: <constructor>(sftp, files, progress)
else
1.1.1.1.6.1: startUpload():void
1.1.1.1.6.1.1: <constructor>(sftp, files, progress)

Figure 12: Sequence diagram of upload function

Figure 11 shows how "Upload" function is implemented. When upload method is called in MainWindow, a confirmation window to ask if user is sure is shown. If user selects yes, upload method from RemoteFileSystem is called. Selected files (array of Files) are given as parameter. RemoteFileSystem then starts the DownloadUploadWindow. First the window is initiated, and instance of TransferProgress is created.

Next one instance of UploadTask, which is an inner class of DownloadUploadWindow, is started. UploadTask is a class which keeps the upload process in a separate thread, so that window can still be updated and cancel button can be pressed while uploading is going on. Inside DownloadTask, a method from J2SSH is used for uploading. This method is given the instance of TransferProgress as parameter, so it then updates the attributes inside the

TransferProgress. Every time attribute has changed in the TransferProgress, PropertyChangedListener launches method in the DownloadUploadWindow. Progress bar and other information about the upload are then updated in GUI.

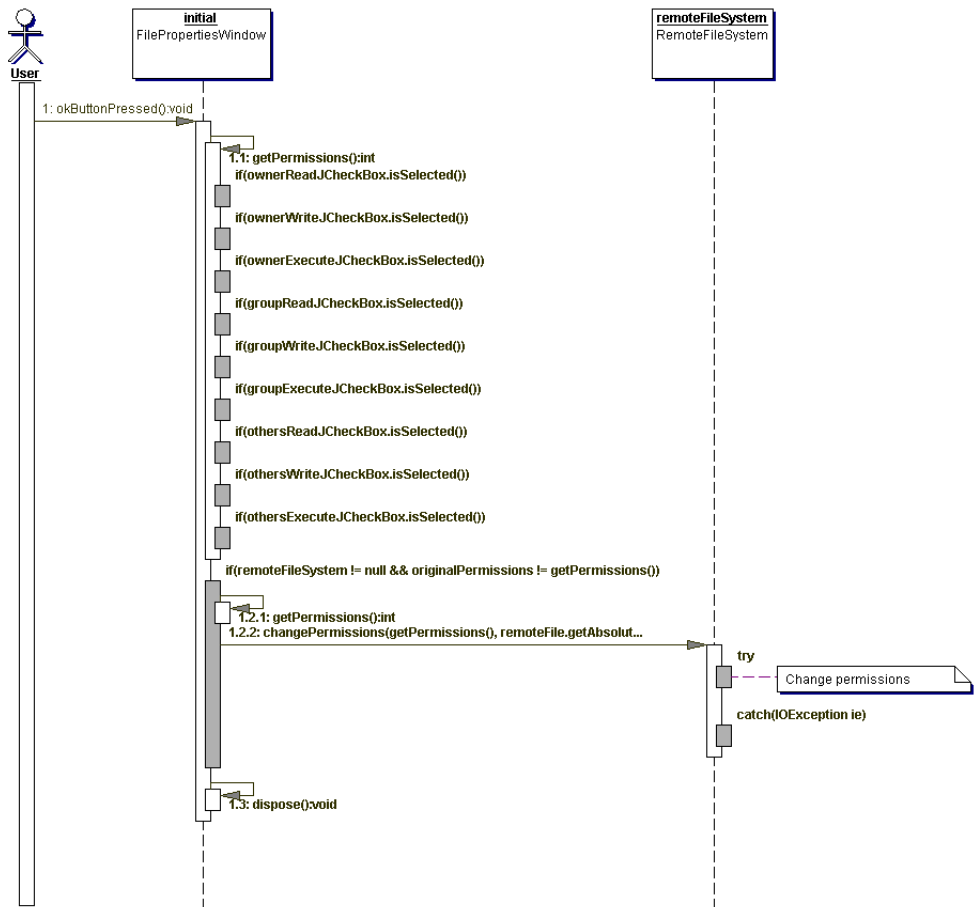

User
initial
FilePropertiesWindow
remoteFileSystem
RemoteFileSystem
1: okButtonPressed():void
1.1: getPermissions():int
if(ownerReadJCheckBox.isSelected())
if(ownerWriteJCheckBox.isSelected())
if(ownerExecuteJCheckBox.isSelected())
if(groupReadJCheckBox.isSelected())
if(groupWriteJCheckBox.isSelected())
if(groupExecuteJCheckBox.isSelected())
if(othersReadJCheckBox.isSelected())
if(othersWriteJCheckBox.isSelected())
if(othersExecuteJCheckBox.isSelected())
if(remoteFileSystem != null && originalPermissions != getPermissions())
1.2.1: getPermissions():int
1.2.2: changePermissions(getPermissions(), remoteFile.getAbsolut...
try
Change permissions
catch(IOException ie)
1.3: dispose():void

Figure 13: Sequence diagram of change file permissions function

Figure 12 shows how “Change file permissions” function is implemented. MainWindow starts the FilePropertiesWindow constructor. When OkButton is pressed (action is performed) in FilePropertiesWindow, the program checks the permission checkboxes in the getPermissions method, and returns an integer representation of the permissions.

Next this integer is compared with another integer which has been initiated at the beginning when the window was created, which represents the original permissions. When at least one of the permissions have changed, changePermissions method of the RemoteFileSystem is called. In there, a method from J2SSH is used to change the permissions.

4 IMPLEMENTATION

In this chapter I will show how some functions are implemented.

4.1 Coding

From the beginning, the design of this program has been done around J2SSH, also known as SSHTools, which is an open source package for Java which provides Java SSH application programming interface (API). It has some ready-made classes and methods which can be used for creating SSH connections and manipulating files through this connection.

4.1.1 Login Function

Snippet 1 has the okButtonPressed method from LoginWindow. At first the values from the GUI text fields are read into variables. A new instance of RemoteFileSystem is then started and these variables are given in as parameters. Next createConnection method is called from the RemoteFileSystem, and the result is saved as String variable. If the result is “connectionOK” string, then main window is opened. Otherwise error is shown.

```
private void okButtonPressed()
{
  String serverAddress = serverAddressJTextField.getText();
  int port = Integer.parseInt(portJTextField.getText());
  String userName = userNameJTextField.getText();
  String password = String.copyValueOf(passwordJPaswordField.getPassword());

  RemoteFileSystem rs = new RemoteFileSystem(serverAddress, port, userName, password);
  String connectionResult = rs.createConnection();
  if (connectionResult.equals("connectionOK"))
  {
    new MainWindow(rs);
    this.dispose();
  }
  else
  {
    errorJLabel.setText(connectionResult);
  }
}
```

Snippet 1: LoginWindow okButtonPressed method

Snippet 2 has the constructor of RemoteFileSystem. All of the login information are saved in variables.

```
public RemoteFileSystem(String serverAddress, int port, String userName, String password)
{
  myServerAddress = serverAddress;
  myUserName = userName;
  myPassword = password;
  myPort = port;
}
```

Snippet 2: RemoteFileSystem constructor

Snippet 3 has the createConnection method. This is where the connection is created. At first the username and password are placed inside pwd which is an instance of PasswordAuthenticationClient (part of J2SSH). A String called connectionResult is created. Next, inside try-block, the connection is created using ssh object which is an instance of SshClient class, also a part of J2SSH. Server address and port is given here as parameters. The third parameter is for the key verification. If some problem happens at this method, an exception is thrown and program jumps to catch-block where proper message is placed in the result string.

```
public String createConnection()
{
  pwd.setUsername(myUserName);
  pwd.setPassword(myPassword);
  String connectionResult = "Unknown error.";

  try
  {
    ssh.connect(myServerAddress, myPort, new IgnoreHostKeyVerification());
    int result = ssh.authenticate(pwd);
    if (result == AuthenticationProtocolState.FAILED)
    {
      connectionResult = "Invalid username or password!";
    }
    else if (result == AuthenticationProtocolState.PARTIAL)
    {
      connectionResult = "Additional authentication required!";
    }
    else if (result == AuthenticationProtocolState.COMPLETE)
    {
      sftp = ssh.openSftpClient();
      connectionResult = "connectionOk";
      homeFolder = sftp.pwd();
    }
  }
  catch (IOException e)
  {
    connectionResult = "Invalid server address or port!";
  }

  return connectionResult;
}
```

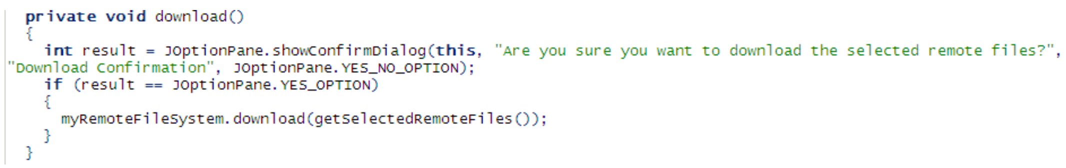
Snippet 3: RemoteFileSystem createConnection method

Normally in SSH, the server's key is saved in local drive and user is required to verify if the key if it has changed. J2SSH unfortunately does not have proper implementation for this functionality, so the verification is ignored and server key is automatically accepted. Next authenticate method of the ssh object is called, to authenticate username and password. This method returns int value which indicates the result. The variable connectionResult is then given a proper text to indicate how the connection went, so the possible error can be shown in LoginWindow or new instance of MainWindow can be opened if connection is ok. In the end, the result string is returned to the LoginWindow where it is used.

4.1.2 Download and Upload Function

As download and upload are very similar, I will only explain download function here.

Snippet 4 shows the download method of MainWindow. In the first line, program calls showConfirmDialog of JOptionPane. The parameters given for this method are parent window (**this** means MainWindow), the message, the title, and last is the options (buttons) shown to user. In this case, yes and no. This method returns the result as integer, and next the program checks if the result is yes. If it is yes, then download method of RemoteFileSystem is called, and the parameter is a vector of SftpFile objects returned by getSelectedRemoteFiles method of MainWindow. I will show the contents of this method next.



```
  private void download()
  {
    int result = JOptionPane.showConfirmDialog(this, "Are you sure you want to download the selected remote files?",
"Download Confirmation", JOptionPane.YES_NO_OPTION);
    if (result == JOptionPane.YES_OPTION)
    {
      myRemoteFileSystem.download(getSelectedRemoteFiles());
    }
  }
```

Snippet 4: MainWindow download method

Snippet 5 shows the getSelectedRemoteFiles method of MainWindow. In first line a new Vector is created. Next an array of integers is received from the myRemoteFileList (JList). This list represents the indices of each selected file in remote file list. This array is for-looped, and for each index, the SftpFile is fetched from the myRemoteFileListModel. It is casted to SftpFile because model returns it as Object. The SftpFile objects are placed to the vector and in the end vector is returned. Now I will show you the download method in RemoteFileSystem.

```
private Vector getSelectedRemoteFiles()
{
  Vector selectedFiles = new Vector();
  int[] selectedIndices = myRemoteFileList.getSelectedIndices();
  for (int i = 0; i < selectedIndices.length; i++)
  {
    selectedFiles.add((SftpFile) myRemoteFileListModel.getElementAt(selectedIndices[i]));
  }
  return selectedFiles;
}
```

Snippet 5: MainWindow getSelectedRemoteFiles method

Snippet 6 shows the download method of RemoteFileSystem. In here, a new instance of DownloadUploadWindow is created. The parameters given into the constructor of DownloadUploadWindow are mainWindow (an instance of MainWindow), sftp (instance of SftpClient, for processing the download), the vector of files, and a boolean true indicating that it is download and not upload.

```
public void download(Vector files)
{
  new DownloadUploadWindow(mainWindow, sftp, files, true);
}
```

Snippet 6: RemoteFileSystem download method

Snippet 7 shows the constructor of DownloadUploadWindow. Program saves the parameters to local variables. Then it calls init() method to initialize window. Next title and icon are set, and window is made visible, and finally startDownload method is called.Snippet 8 shows the startDownload method. In this method, program creates a new instance of DownloadTask and calls the execute method of it. Parameters of SftpClient, vector of files and instance of TransferProgress is given in. DownloadTask is a class which extends SwingWorker class. This is used to keep the download process in a separate thread.

```
public DownloadUploadWindow(MainWindow mainWindow, SftpClient sftp, Vector files, boolean isDownload)
{
  this.mainWindow = mainWindow;
  this.sftp = sftp;
  this.files = files;
  this.isDownload = isDownload;
  init();
  if (isDownload == true)
  {
    this.setTitle("Download");
    this.setIconImage(new ImageIcon(getClass().getResource("images/Earth-Download.png")).getImage());
  }
  else
  {
    this.setTitle("Upload");
    this.setIconImage(new ImageIcon(getClass().getResource("images/Earth-Upload.png")).getImage());
  }
  this.setVisible(true);

  if (isDownload == true)
  {
    startDownload();
  }
  else
  {
    startUpload();
  }
}
```

Snippet 7: DownloadUploadWindow constructor

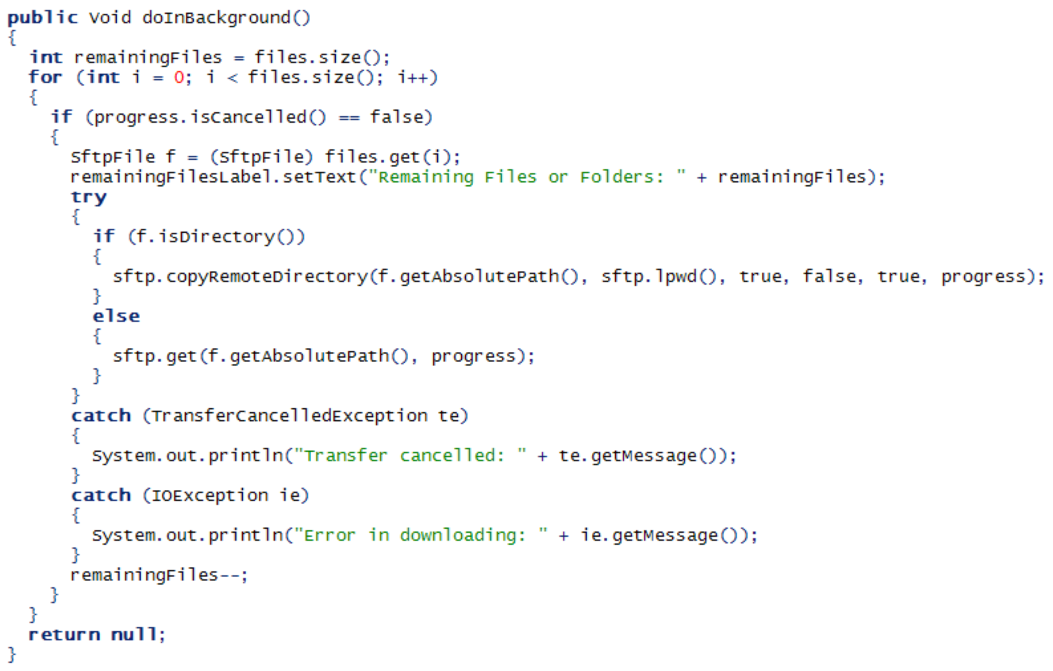
Snippet 8 shows the startDownload method. In this method, program creates a new instance of DownloadTask and calls the execute method of it. Parameters of SftpClient, vector of files and instance of TransferProgress is given in. DownloadTask is a class which extends SwingWorker class. This is used to keep the download process in a separate thread.

```
private void startDownload()
{
  downloadTask = new DownloadTask(sftp, files, progress);
  downloadTask.execute();
}
```

Snippet 8: DownloadUploadWindow startDownload method

Snippet 9 shows doInBackground method of DownloadTask. This is a method where the actual task of SwingWorker should be placed. In this method, program for-loops through the vector of files. For each file, it is checked if it is a directory or a file. If it is a directory, then copyRemoteDirectory method of SftpClient is used. If it is a file, then get method is used. The instance of TransferProgress is given here as parameter. As TransferProgress implements a J2SSH interface, it can be used here. PropertyChangeListener listens the changes of variables in

TransferProgress and whenever change is made by the J2SSH to the TransferProgress, a propertyChange method is called in DownloadUploadWindow.



```java
public void doInBackground()
{
  int remainingFiles = files.size();
  for (int i = 0; i < files.size(); i++)
  {
    if (progress.isCancelled() == false)
    {
      SftpFile f = (SftpFile) files.get(i);
      remainingFilesLabel.setText("Remaining Files or Folders: " + remainingFiles);
      try
      {
        if (f.isDirectory())
        {
          sftp.copyRemoteDirectory(f.getAbsolutePath(), sftp.lpwd(), true, false, true, progress);
        }
        else
        {
          sftp.get(f.getAbsolutePath(), progress);
        }
      }
      catch (TransferCancelledException te)
      {
        System.out.println("Transfer cancelled: " + te.getMessage());
      }
      catch (IOException ie)
      {
        System.out.println("Error in downloading: " + ie.getMessage());
      }
      remainingFiles--;
    }
  }
  return null;
}
```

Snippet 9: DownloadTask doInBackground method

Snippet 10 shows the propertyChange method of DownloadUploadWindow. At first, it is checked that the source is equal to the instance of TransferProgress. It actually always is, as the listener is only placed on it. The second if checks if the progress has not still started, and makes the progress bar to be in intermediate mode. This makes a small bar to go from left to right, to indicate that it is waiting for progress to start.

```
public void propertyChange(PropertyChangeEvent evt)
{
  if (evt.getSource().equals(progress))
  {
    if (progress.getBytesSoFar() == 0)
    {
      progressBar.setIndeterminate(true);
    }
    else
    {
      currentFileLabel.setText("File: " + progress.getFileName());
      currentFileSizeLabel.setText("Size: " + getFileSize(progress.getTotal()));
      progressBar.setIndeterminate(false);
      progressBar.setMinimum(0);
      progressBar.setMaximum((int) progress.getTotal());
      progressBar.setValue((int) progress.getBytesSoFar());
    }
  }
}
```

Snippet 10: DownloadUploadWindow propertyChange method

If the progress has started, the information is updated in the GUI. File name, size, and progress so far are all variables in the TransferProgress and are fetched using methods inside it.

Snippet 11 shows the done method of DownloadTask. SwingWorker goes to this method after the task of doInBackground method has finished. In this method, Progress bar is set to 100%, program sets the cancel button disabled and ok button enabled. The other labels are cleared and “Download Complete!” text is shown in one of them.

```
public void done()
{
  progressBar.setValue(progressBar.getMaximum());
  okJButton.setEnabled(true);
  cancelJButton.setEnabled(false);
  remainingFilesLabel.setText(" ");
  currentFileLabel.setText("Download Complete!");
  currentFileSizeLabel.setText(" ");
}
```

Snippet 11: DownloadTask done method

4.1.3 Rename Function

Snippet 12 shows the rename method of MainWindow. In this method, program first checks if the local or remote is selected (LOCAL and REMOTE are constant integer variables), and then starts a new instance of RenameWindow. After the RenameWindow has been closed, refresh method, which updates the file lists shown in main window, is called. The parameters given in each case are different so they both use different constructors of RenameWindow.

```
private void rename()
{
  if (selectedSystem == LOCAL)
  {
    new RenameWindow(this, myLocalFileSystem, (File) myLocalFileList.getSelectedValue());
    refresh();
  }
  else if (selectedSystem == REMOTE)
  {
    new RenameWindow(this, myRemoteFileSystem, (SftpFile) myRemoteFileList.getSelectedValue());
    refresh();
  }
}
```

Snippet 12: MainWindow rename method

Snippet 13 shows the constructor of RenameWindow which is used when renaming remote files. Parameters are instance of MainWindow, instance of RemoteFileSystem and instance of SftpFile which represents the file or folder that is going to be renamed. RenameWindow extends JDialog class, so the constructor of super is called. The name of the file is placed into a text field in the window, where user can change it.

```
public RenameWindow(JFrame mainWindow, RemoteFileSystem rfs, SftpFile file)
{
  super(mainWindow, true);
  myRemoteFileSystem = rfs;
  remoteFile = file;
  renameTextField.setText(file.getFilename());
  selectedSystem = REMOTE;
  init();
}
```

Snippet 13: RenameWindow constructor (remote)

When user presses OK, program starts okButtonPressed method from actionPerformed. The first "if" is just checking that the text field is not empty.

Next, the renameFile method of LocalFileSystem or RemoteFileSystem is called depending on if the file was local or remote.

```java
private void okButtonPressed()
{
  if (!renameTextField.getText().trim().equals(""))
  {
    if (selectedSystem == LOCAL)
    {
      myLocalFileSystem.renameFile(localFile, renameTextField.getText());
      this.dispose();
    }
    else if (selectedSystem == REMOTE)
    {
      try
      {
        myRemoteFileSystem.renameFile(remoteFile.getFilename(), renameTextField.getText());
        this.dispose();
      }
      catch (Exception e)
      {
        JOptionPane.showMessageDialog(this, "Error!  ");
        System.out.println("Error. Can not rename file or folder!");
      }
    }

  }
}
```

Snippet 14: RenameWindow okButtonPressed method

Snippet 15 shows the LocalFileSystem's renameFile method. At first, an instance of File is created with the new name that is given to the method as parameter. If this file exists, a message is shown to the user. If not, then renameTo method from an instance of File, which represents the original file and was passed to this method as parameter, is called. This method requires parameter that is an instance of File that represents the new name.

```java
public void renameFile(File file, String newName)
{
  try
  {
    File f = new File(currentFolder, newName);
    if (f.exists())
    {
      JOptionPane.showMessageDialog(null, "A file or folder with the given name already exists!");
    }
    else
    {
      file.renameTo(f);
    }
  }
  catch (Exception ie)
  {
    JOptionPane.showMessageDialog(null, "Error. Unable to rename!");
  }
}
```

Snippet 15: LocalFileSystem renameFile method

Snippet 16 shows the rename method of RemoteFileSystem. This differs somewhat from the LocalFileSystem, as this one uses J2SSH and not classes from java.util package. Parameters for this method are just the old and new file name. A rename method from SftpClient is called and old name and new name are also given as parameters to it. If there is problem with renaming, exception is thrown and message dialog in catch-block informs the user about it.

```
public void renameFile(String oldName, String newName)
{
  try
  {
    sftp.rename(oldName, newName);
  }
  catch (IOException e)
  {
    JOptionPane.showMessageDialog(null, "Error. Unable to rename! " + e.getMessage());
    System.out.println(e.getMessage());
  }
}
```

Snippet 16: RemoteFileSystem rename method

4.1.4 Delete

Snippet 17 shows the delete function of MainWindow. This is very similar to the download function. At first, user is asked a confirmation through JOptionPane's showConfirmationDialog method. Then deleteFiles method is started from LocalFileSystem or RemoteFileSystem (depending on which is selected). Next the file lists are refreshed.

```
  private void delete()
  {
    if (selectedSystem == LOCAL)
    {
      int result = JOptionPane.showConfirmDialog(this, "Are you sure you want to delete the selected local files?",
"Delete Confirmation", JOptionPane.YES_NO_OPTION);
      if (result == JOptionPane.YES_OPTION)
      {
        myLocalFileSystem.deleteFiles(getSelectedLocalFiles());
        refresh();
      }
    }
    else if (selectedSystem == REMOTE)
    {
      int result = JOptionPane.showConfirmDialog(this, "Are you sure you want to delete the selected remote files?",
"Delete Confirmation", JOptionPane.YES_NO_OPTION);
      if (result == JOptionPane.YES_OPTION)
      {
        myRemoteFileSystem.deleteFiles(getSelectedRemoteFiles());
        refresh();
      }
    }
  }
```

Snippet 17: MainWindow delete function

For remote system, deleting is quite straightforward. At first, the vector of selected files is for-looped through, and rm method of SftpClient is called. The parameters given are the path of the file, and one of the booleans that is true means that it should delete recursively (all files and folders under one folder). If problem appears, the user is informed with message dialog.

```java
public void deleteFiles(Vector files)
{
  try
  {
    for (int i = 0; i < files.size(); i++)
    {
      sftp.rm(((SftpFile) files.get(i)).getAbsolutePath(), true, true);
    }
  }
  catch (IOException e)
  {
    JOptionPane.showMessageDialog(null, "Error in deleting files! " + e.getMessage());
  }
}
```

Snippet 18: RemoteFileSystem deleteFiles method

For local system, deleting is not as easy, because Java does not include methods for deleting folders recursively. While the files are for-looped, another method deleteRecursively is called. This deleteRecursively method takes care of deleting all subfolders and files in case it is a folder.

```java
public void deleteFiles(Vector files)
{
  try
  {
    // Looping through each file/folder in the vector and deleting them
    for (int i = 0; i < files.size(); i++)
    {
      boolean result = deleteRecursively((File)files.get(i));
      if (result == false)
      {
        JOptionPane.showMessageDialog(null, "Error. Unable to delete file: " + ((File)files.get(i)).getName());
      }
    }
  }
  catch (Exception e)
  {
    JOptionPane.showMessageDialog(null, "Error in deleting files! ");
  }
}
```

Snippet 19: LocalFileSystem deleteFiles method

Snippet 20 shows deleteRecursively method, which is a recursive method (meaning that this method itself is called inside of it). At first it checks if the file exists. If it does not exist, it returns true, meaning delete successful (as file does not exists.

```
private boolean deleteRecursively(File file)
{
  if (!file.exists())
  {
    return true;
  }

  boolean result = true;
  if (file.isDirectory())
  {
    File[] files = file.listFiles();
    for (int i = 0; i < files.length; i++)
    {
      result &= deleteRecursively(files[i]);
    }
    result = file.delete();
  }
  else
  {
    result = file.delete();
  }
  return result;
}
```

Snippet 20: LocalFileSystem deleteRecursively method

Next, it checks if the file is directory or not. If it is a directory, the files/folders under this directory are fetched into array of File objects, and each of them is deleted using the method itself. After the for loop, the folder itself is deleted. If it is not a directory, file is simply deleted with delete method of Java's File class.

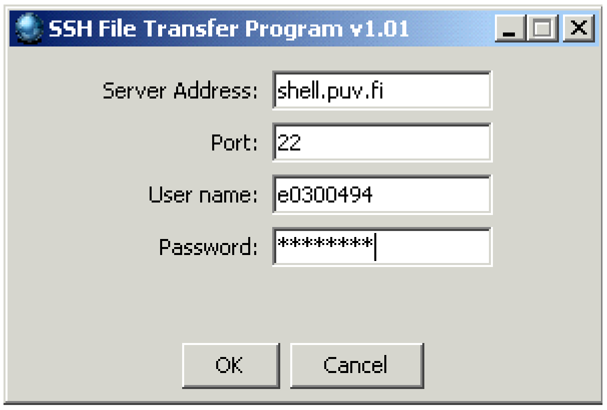
4.1.5 Testing

Testing is an important part of implementation of any software. With testing, all possible errors in logic and bugs can be found. Testing of this program was done with two different operating systems: Windows and Linux. In addition to my personal testing during the coding of this program, I also sent versions of it to my friends who were testing with their own computers and gave me feedback which I then used to improve the application.

4.2 Results

In the results and conclusions of this project, I show you how the program works and looks like, and I also give some personal opinions and conclusions about it. I also explain something about the future work that can be done with this project. In this chapter, I show and explain how each function of this program works.

4.2.1 Login Window



Server Address: shell.puv.fi
Port: 22
User name: e0300494
Password: ********

Figure 14: Login window

Figure 14 shows the login window. In the title, it has the name of the program and version. Below there are 4 labels and 4 text fields. This is where the user gives in the information about login. If one of the text fields are empty, OK button becomes disabled and can not be pressed. If user presses OK button, or presses enter in one of the text fields, the connection is established and main window opened. If there is problem in the connection, error message is shown above the buttons, below the text fields, and main window is not opened.

4.2.2 Main Window

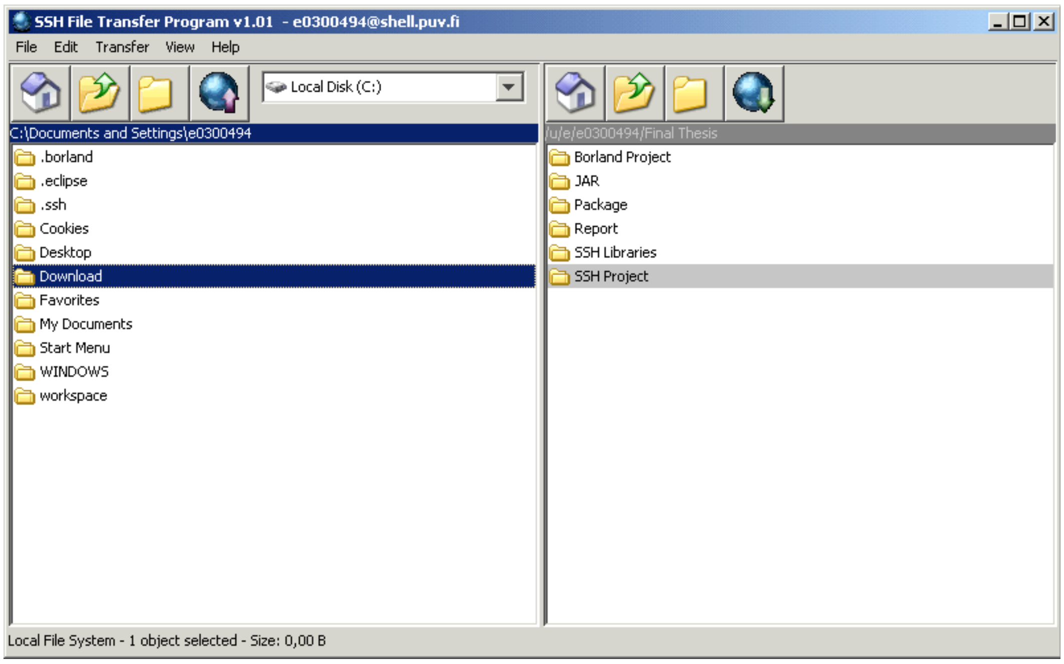

SSH File Transfer Program v1.01 - e0300494@shell.puv.fi
File Edit Transfer View Help
Local Disk (C:)
C:\Documents and Settings\e0300494
.borland
.eclipse
.ssh
Cookies
Desktop
Download
Favorites
My Documents
Start Menu
WINDOWS
workspace
/u/e/e0300494/Final Thesis
Borland Project
JAR
Package
Report
SSH Libraries
SSH Project
Local File System - 1 object selected - Size: 0,00 B

Figure 15: Main Window

Main window, as seen in Figure 15, is divided from middle in two main parts. The local file list is on the left side and remote file list is on the right side. Above each file list there is tool bar for each list. The buttons in the tool bar are Home, Parent Folder, New Folder and Upload/Download. On the local file list, there is also selection for drive.

Above the tool bar there is menu, which has following items. File (Create New Folder, Properties, Exit), Edit (Rename, Delete), Transfer (Upload, Download), View (Show Hidden Files, Refresh), Help (About). Most of these menu items are self-explanatory. At the bottom of the window, there is status bar which shows the selected file system (local or remote) and number of selected objects and their size. Size is only shown for files, as looking inside folders for size would take too long in some cases.

4.2.3 Popup Menu

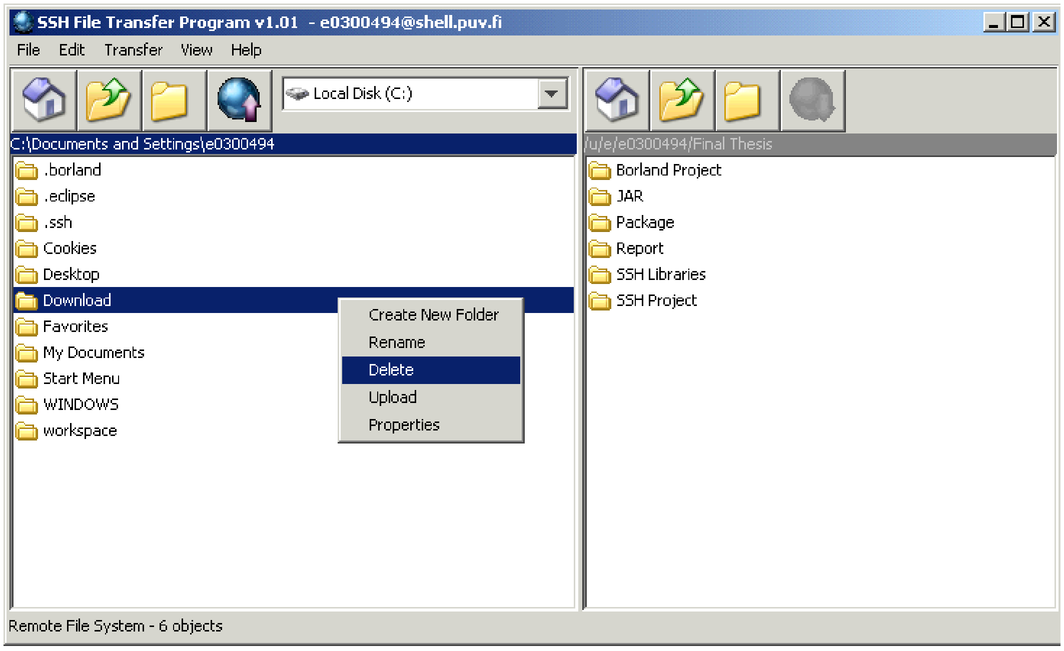


Figure 16: Popup menu

There is also a popup menu, which appears when right mouse button is clicked. This menu has following items: Create New Folder, Rename, Delete, Upload/Download, Properties. These items, apart from Create New Folder, are only available when file(s) or folder(s) are selected. Figure 16 shows how the popup menu looks like.

4.2.4 Show Hidden Files

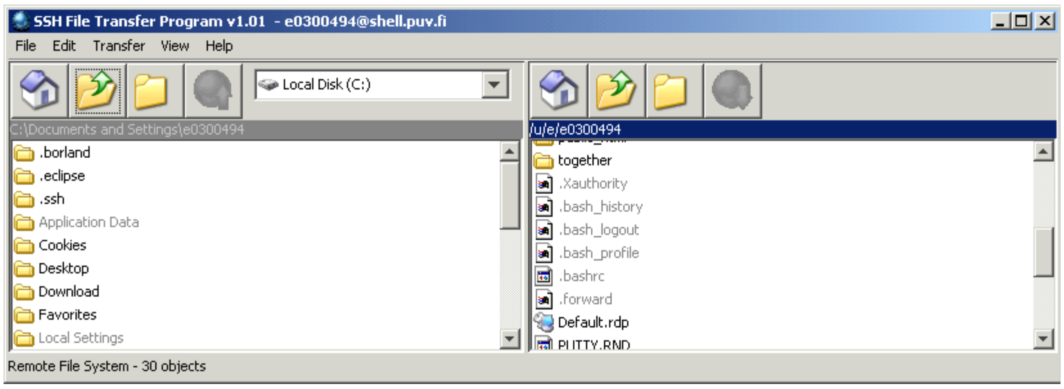
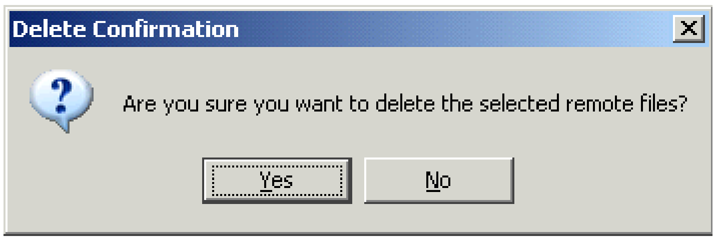


Figure 17: Show hidden files

When Show Hidden Files menu item has been selected from the View menu, files that are normally hidden are also visible. These files are shown with grey color instead of black, to let the user see which files are hidden and which are not hidden.

4.2.5 Deleting Files



Are you sure you want to delete the selected remote files?

Figure 18: Delete confirmation

Deleting files can be done through menu items or with delete key. When deleting files, a confirmation dialog appears. This window is not created from a class in the ui package, but is instead created using Java's default JOptionPane. Many other confirmations, such as confirmation for downloading and uploading files are done with the same way.

4.2.6 Renaming Files

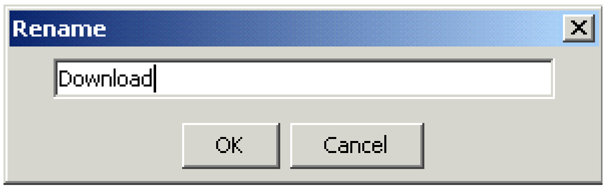


Figure 19: Rename window

When choosing rename from the menu, or by pressing F2, a rename window will appear. This window has a text field for the user to give the new name for the file, and OK and Cancel buttons for actions. When the window opens at first, the current name for the file or folder is shown in the text field. When user is satisfied for the new name, he or she can press OK button. If he or she wants to cancel, it can be done easily with Cancel button.

4.2.7 Creating New Folders

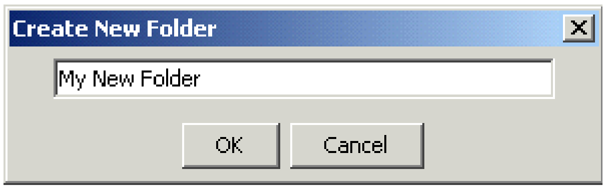
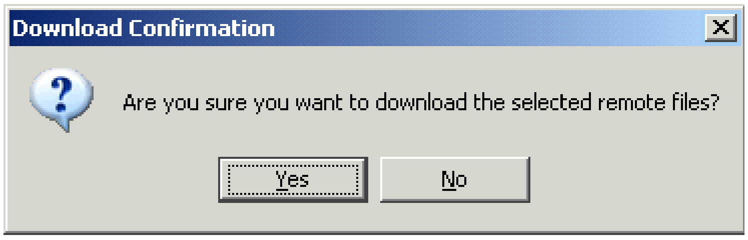


Figure 20: Create New Folder window

Creating new folders, which can be done through menu or through the button on tool bar, is as simple as renaming files or folders. User simply needs to give the name for the new folder and click OK. Cancelling the operation can again be easily done with Cancel button.

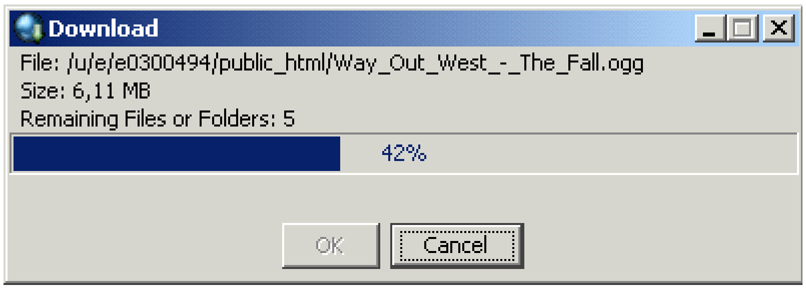
4.2.8 Downloading and Uploading



Are you sure you want to download the selected remote files?

Figure 21: Download confirmation

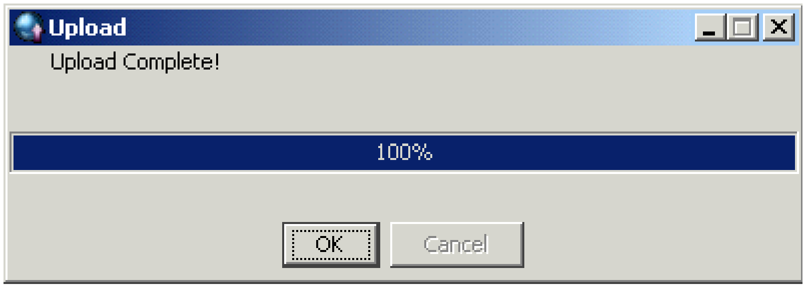
When the user initiates download through the tool bar button or menus, a confirmation dialog appears first to make sure that user really wants to download the selected files or folders. If user presses Yes, the download is started and progress window is shown.



File: /u/e/e0300494/public_html/Way_Out_West_-_The_Fall.ogg
Size: 6,11 MB
Remaining Files or Folders: 5

Figure 22: Download progress window

In the Download progress window (Figure 22), user can see the name and location of the current file, the size of the current file and number of remaining files or folders. The progress is shown with a progress bar and user can cancel the download at any moment with Cancel button.


Upload

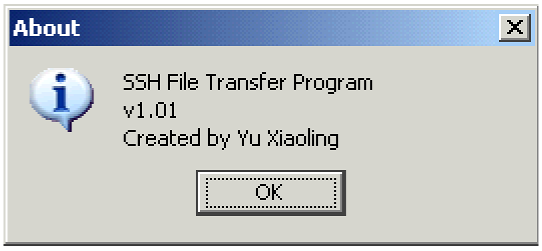
Upload Complete!

100%

Figure 23: Upload progress window - upload completed

Uploading is done in the same way as downloading. When the process has finished, a text shows that download/upload has completed and OK button becomes active. When user presses OK button, the window closes.

4.2.9 About Window


About

SSH File Transfer Program
v1.01
Created by Yu Xiaoling

Figure 24: About window

When user selects About from the Help menu, About window is shown. About window has the program's name, version and author. This window has also been implemented with JOptionPane (as are the confirmations), since it is just used to show text.

5 CONCLUSIONS

This project has carried out the purpose to transfer files from local system to remote system with SSH connection, and vice versa. The project work has fulfilled all of the requirements. It works well so that all the functions are implemented and also it offers user-friendliness which makes it easier for users to use compared with more complex programs.

It will definitely help users to transfer files easily with the SSH connection. For most of the functions, users have multiple ways to access them. Take download and upload as example: User can access them from menu, tool bar and right-click popup menu. Hence, users can choose the way they like to approach their goal.

From my point of view, this project has been very beneficial for me. The project has been very challenging, but it has also helped me to learn a lot more about Java programming. One of the biggest challenge has been the fact that SSHTools did not include proper documentation. The in-code documentation (javadoc) was completely missing, and even the separate documentation was incomplete. This forced me to look more deeply inside the source code of SSHTools, which gave me deeper knowledge of how it works.

Besides, I have also seen how powerful the internet search engines are. Whenever I meet problems or challenges, I search the relevant topics from the search engine and it always finds me pages from where I can find the way towards the solution. In fact, sometimes I can see others having similar questions and good answers for them by experienced users, which is important and beneficial for me as well.

For the future, there are still some fields that can be improved in this program. Some of them are implementation of "drag and drop", where user could be able to just drag the files or folders between local and remote and initiate upload or download in that way. The properties for local file system could also be more versatile, including the read-only option and selecting the hidden status.

As a conclusion, the project has benefited me since it has developed my programming skills, as well as other people who will be users of this program in the future.

6 SUMMARY

The aim of my final thesis project work was to develop a program with which user is able to transfer files from local computer to a server, and vice versa, with SSH protocol. Moreover, it offers simple and user friendly interface so that user can use this program easily. This project was done in these phases: information gathering, analysis, design and implementation.

I have used Java Swing technology to code my project. Besides, an open source SSH package, SSHTools (J2SSH) was used for the implementation of the SSH part of this program. I have chosen Eclipse as the IDE (Integrated Development Environment) to create the application, as a fact, I have found that Eclipse is a powerful tool to develop Java application.

To develop this project was a big challenge for me. Although, I am still happy to see how the project meets my expectations. Most important is that this project will benefit potential future users. If possible, I would like to continue to study more about Java Swing so that “drag and drop” function would work.

LIST OF REFERENCES

[1] Learn About Java Technology http://www.java.com/en/about/

[2] What is Swing? http://java.sun.com/docs/books/tutorial/ui/overview/intro.html

[3] Java Look and Feel Guidelines http://java.sun.com/products/jlf/ed2/book/

[4] Webopedia – What is SSH? http://www.webopedia.com/TERM/S/SSH.html

[5] Wikipedia – SSH File Transfer Protocol
http://en.wikipedia.org/wiki/SSH_file_transfer_protocol

[6] WinSCP - SFTP http://winscp.net/eng/docs/protocols#sftp

[7] SSHTools http://sourceforge.net/projects/sshtools

[8] The Java Tutorials – How to Use SpringLayout
http://java.sun.com/docs/books/tutorial/uiswing/layout/spring.html

[9] Sun Microsystems - Model-View-Controller Image
http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/images/app-archa2.gif