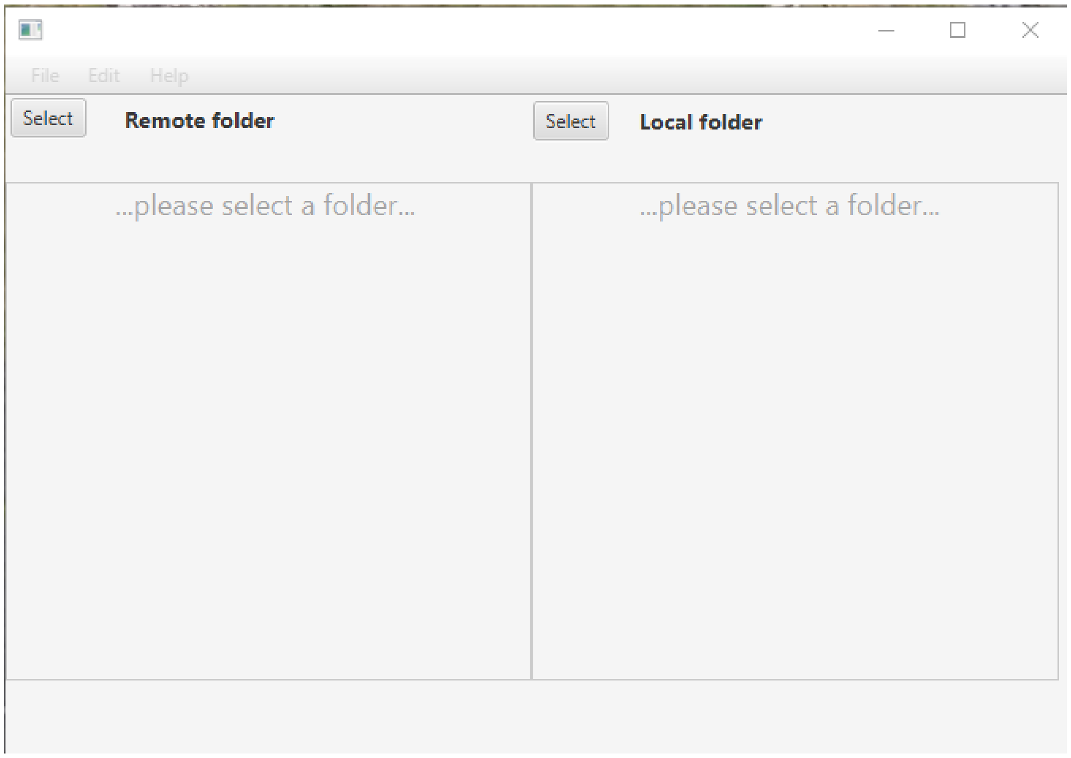
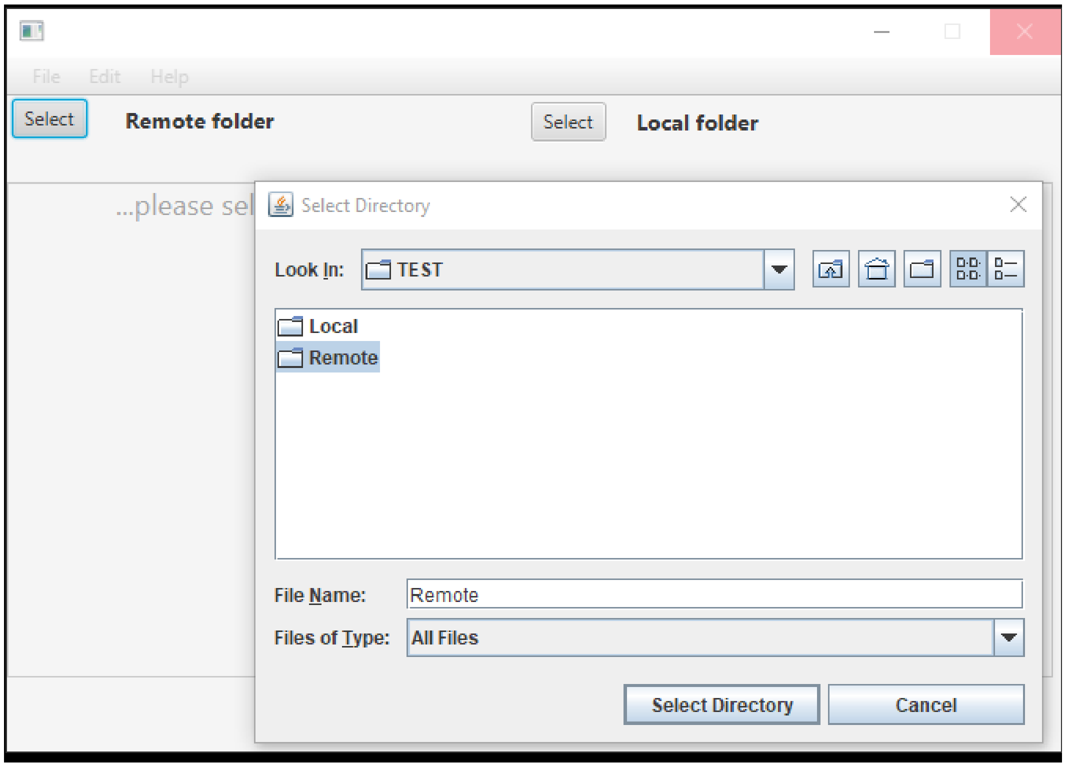
# Functionality

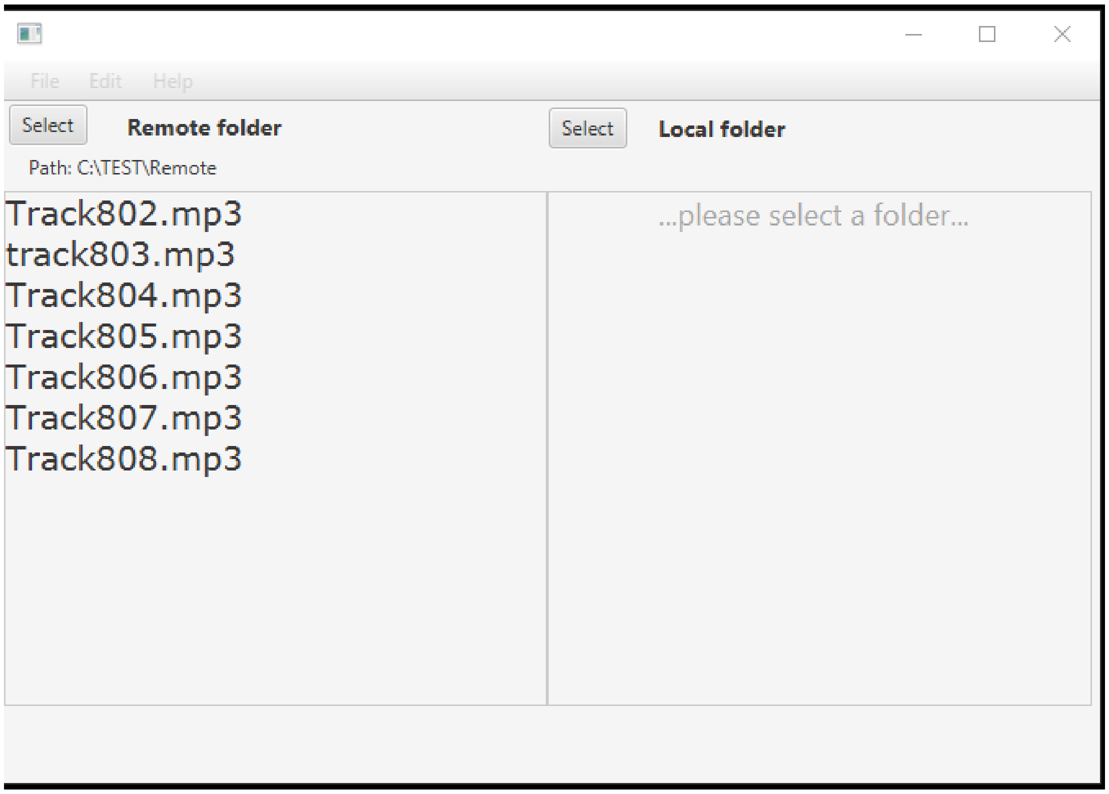
On startup, we can choose the remote and local folder.



Buttons use file selectors.

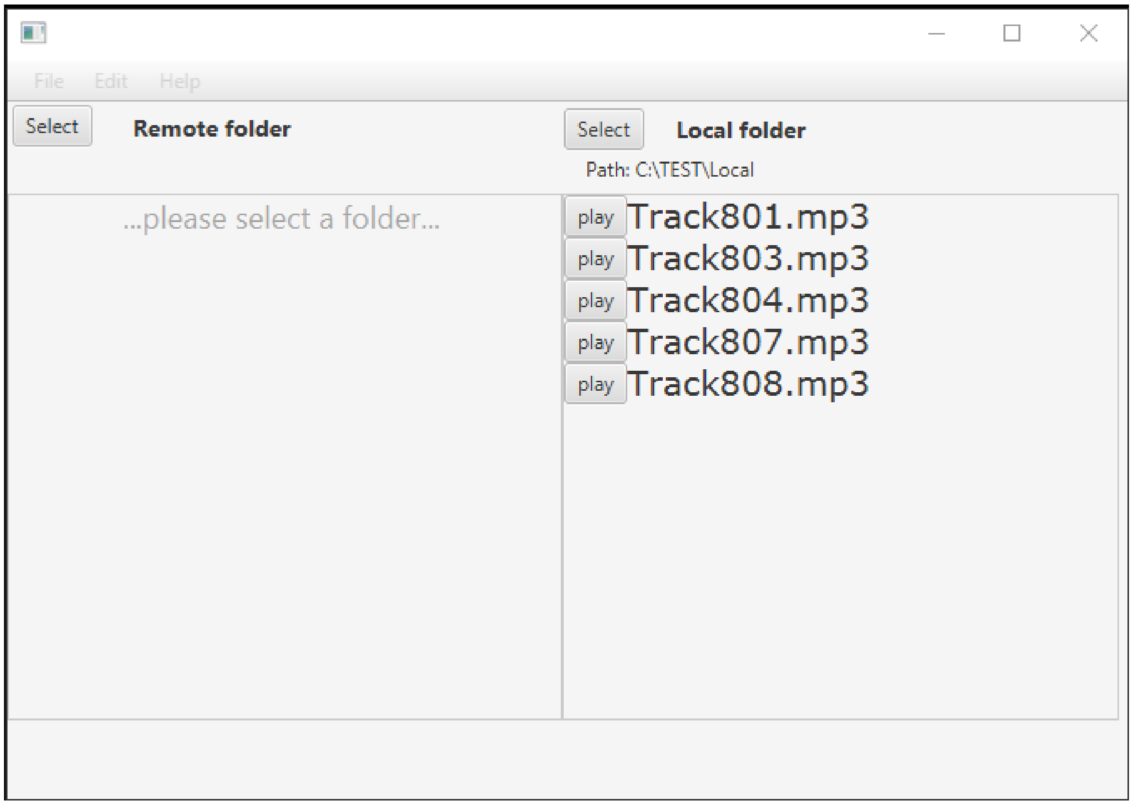


Example of the Remote folder when loaded.



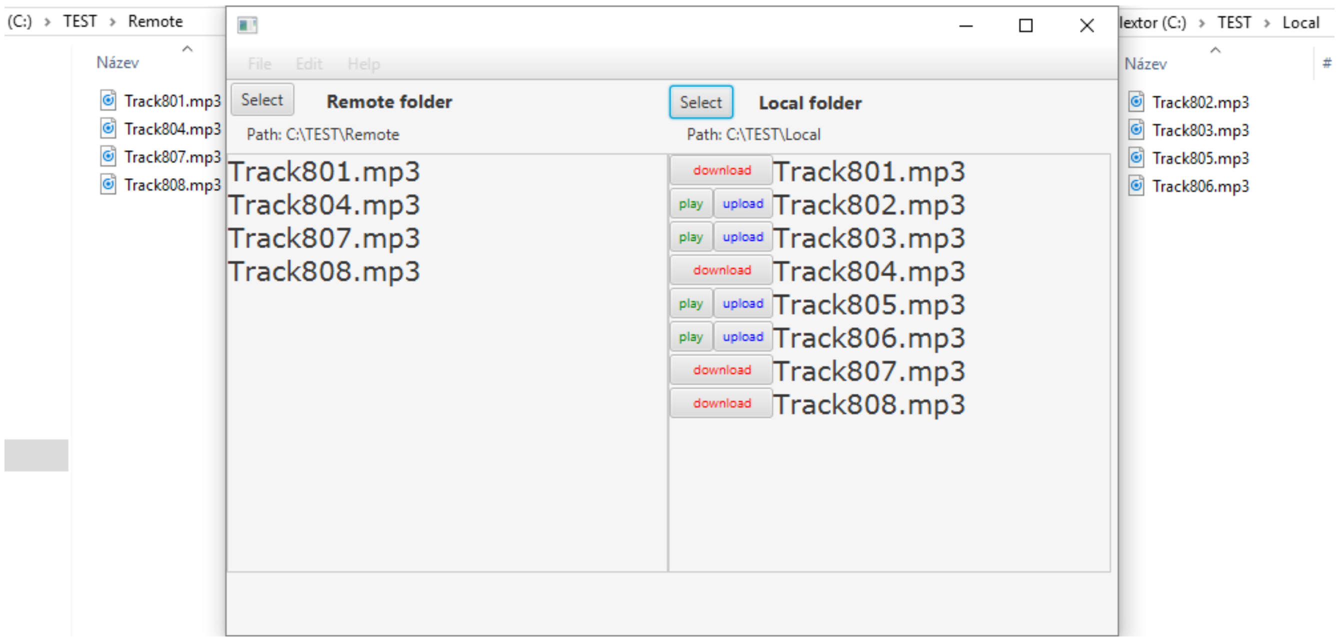
Track802.mp3
track803.mp3
Track804.mp3
Track805.mp3
Track806.mp3
Track807.mp3
Track808.mp3

If the local folder is selected first, we have only "play" button available, because the remote folder is not defined.

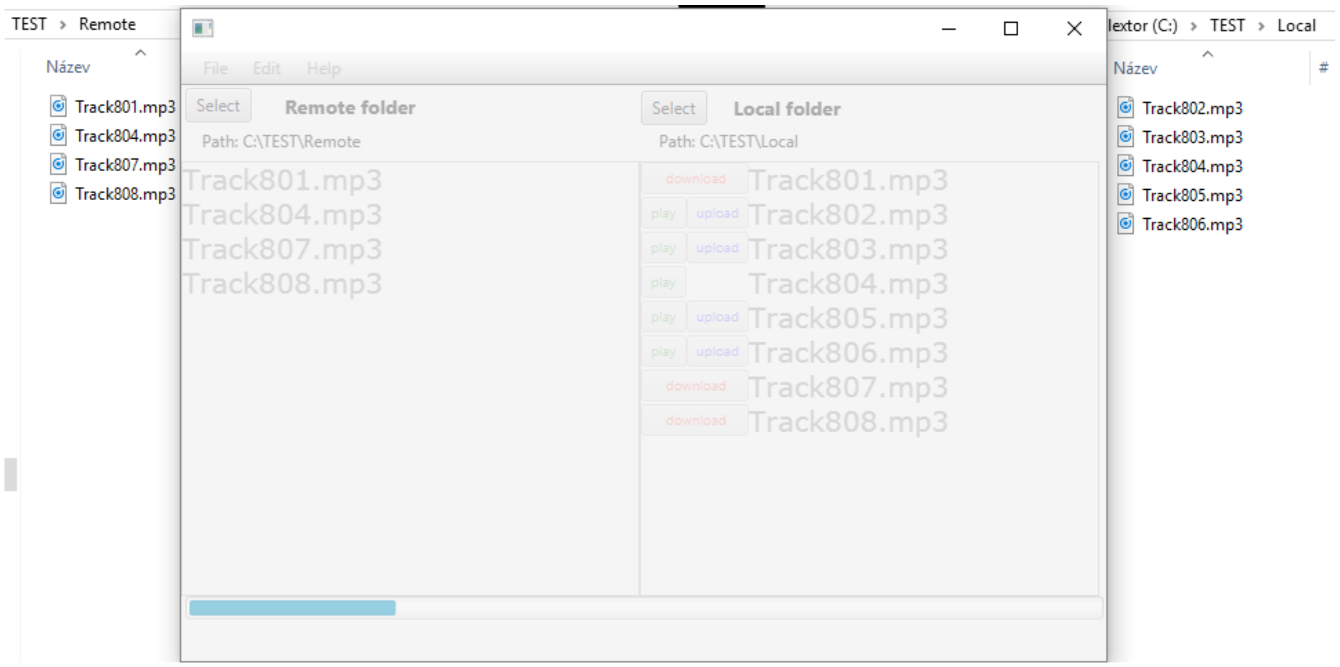


Track801.mp3
Track803.mp3
Track804.mp3
Track807.mp3
Track808.mp3

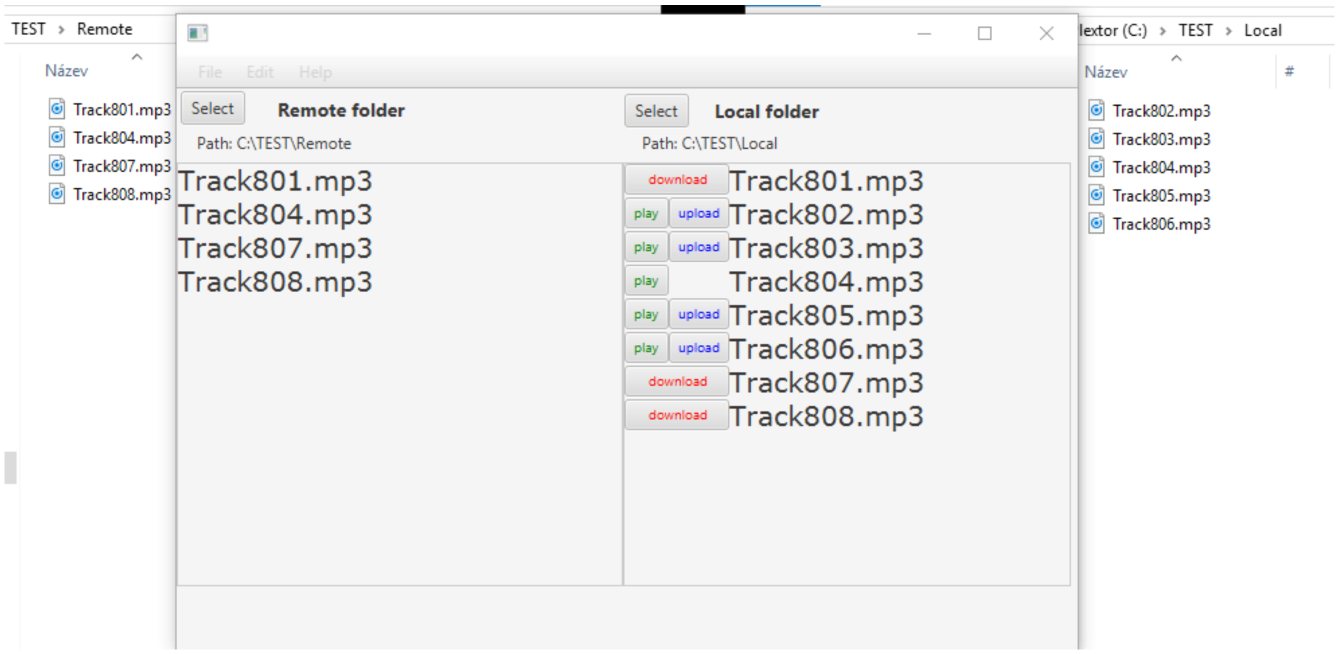
Application with real folders. Remote on left, local on the right.



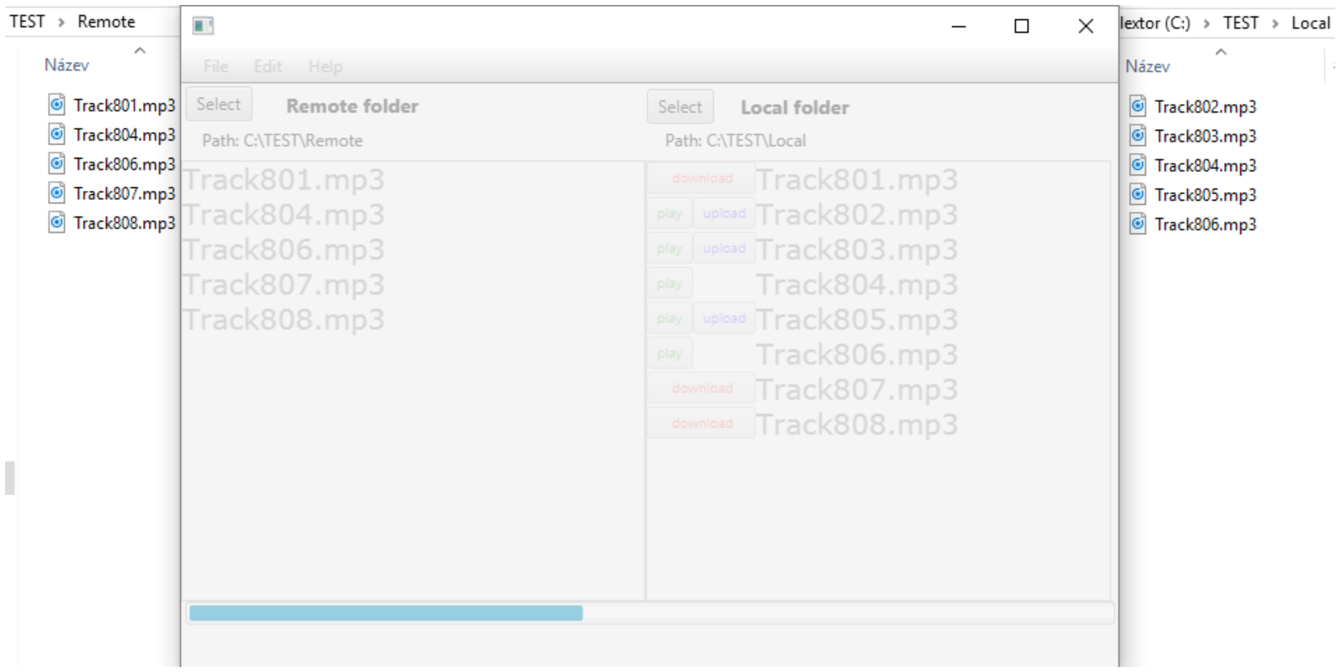
When the download button is pressed, all buttons are disabled for safety reason and ProgressBar showing the progress. The length of the bar is the size of the file. Each step is one Byte. Done in Task.



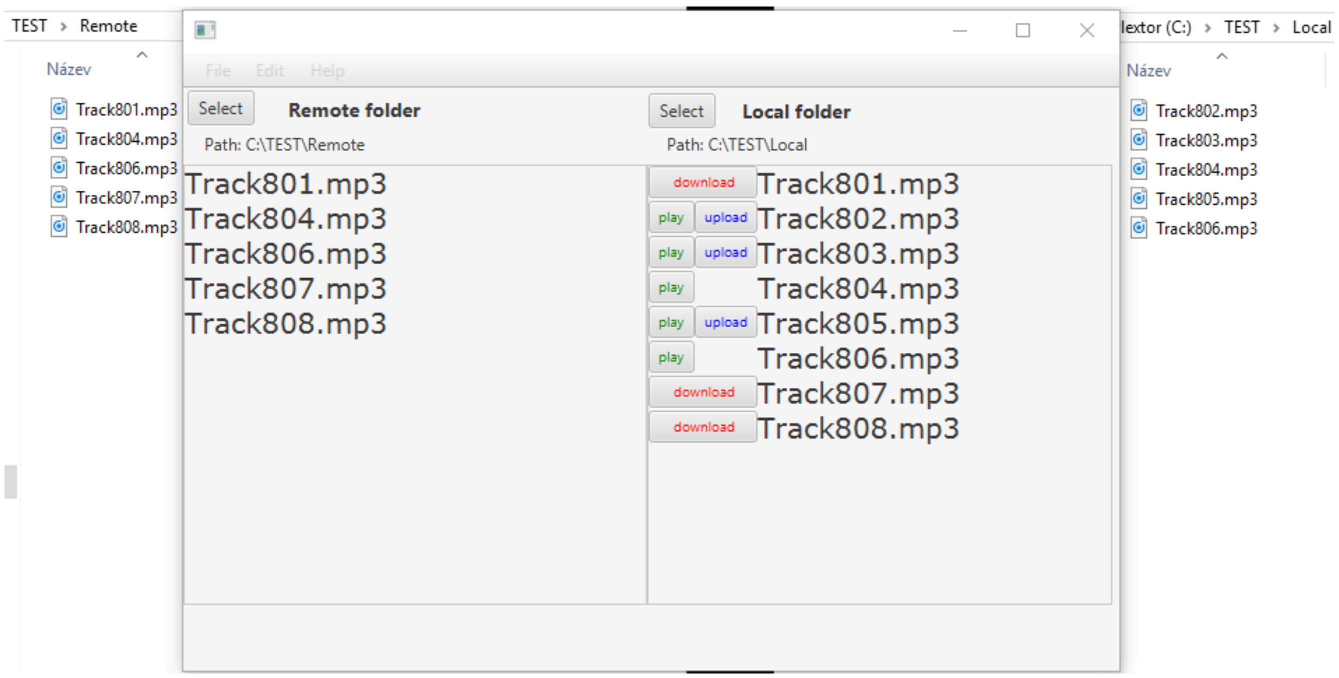
When the download is finished, the download button turns to play button and all buttons are enabled.



Upload works same as the download but another way.



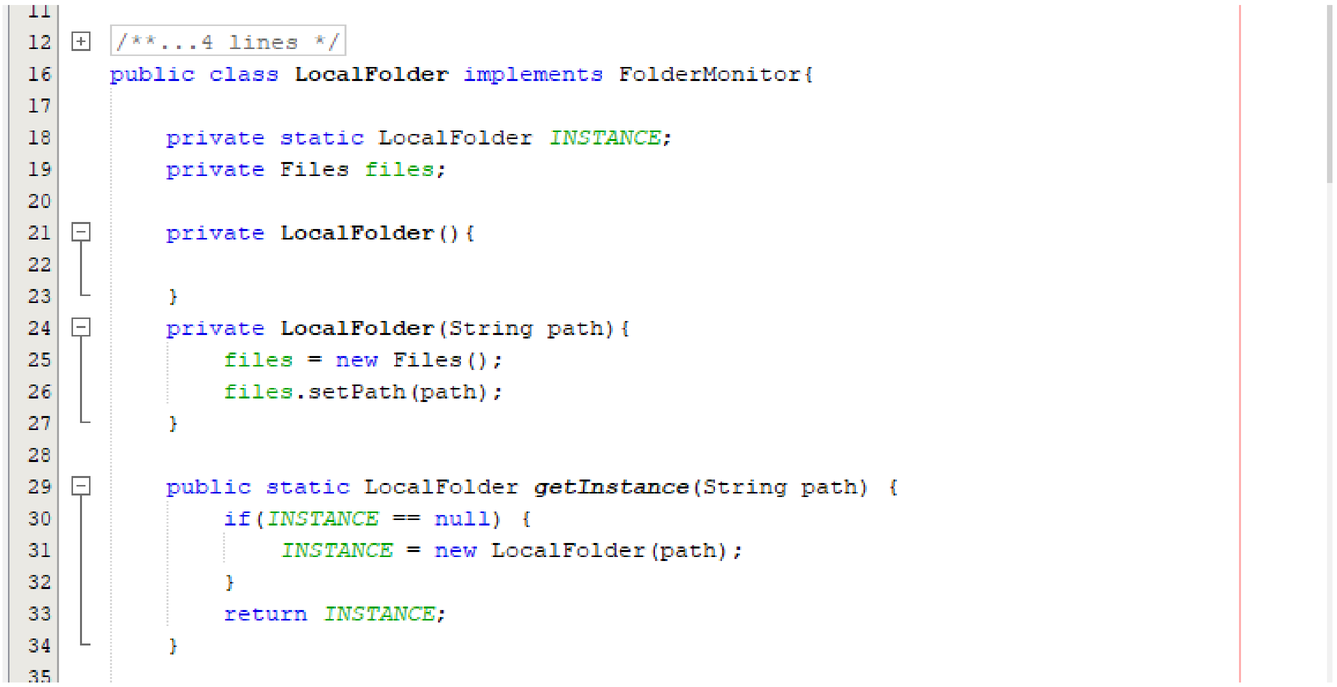
When upload finish, everything is enabled.



## Design and Patterns

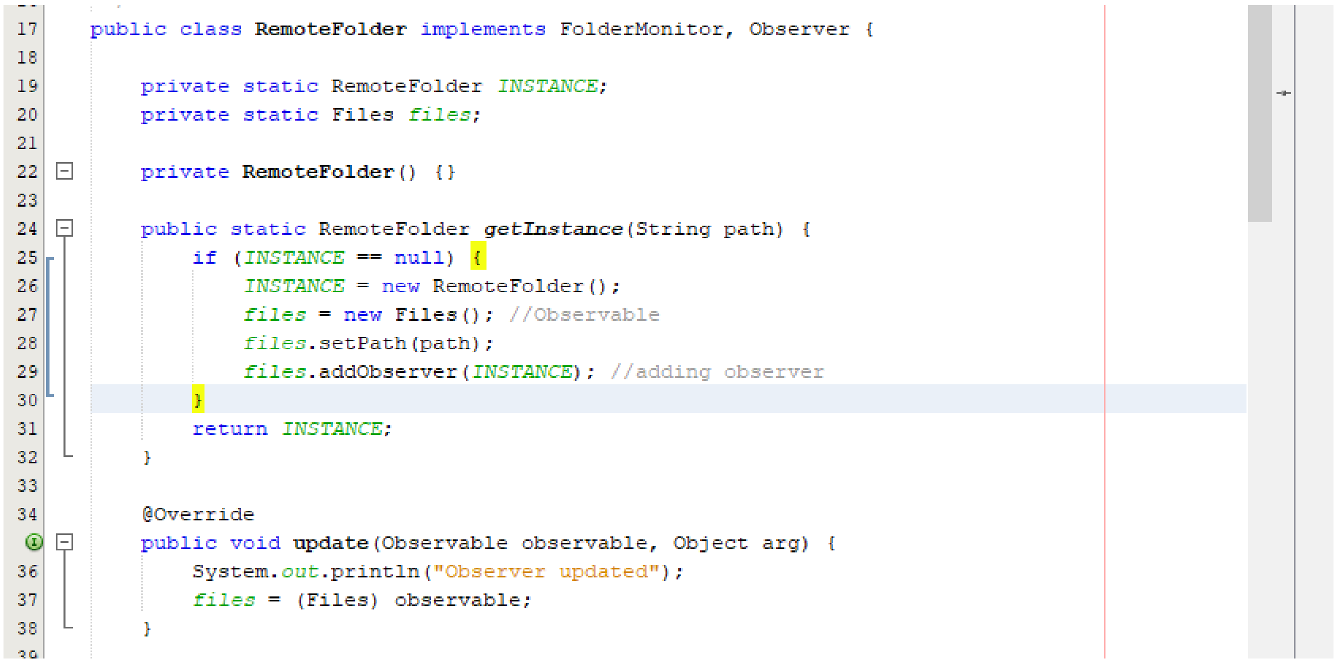
### Singleton pattern

is used in local folder class.



```
/**...4 lines */
public class LocalFolder implements FolderMonitor{

    private static LocalFolder INSTANCE;
    private Files files;

    private LocalFolder(){

    }
    private LocalFolder(String path){
        files = new Files();
        files.setPath(path);
    }

    public static LocalFolder getInstance(String path) {
        if(INSTANCE == null) {
            INSTANCE = new LocalFolder(path);
        }
        return INSTANCE;
    }
```

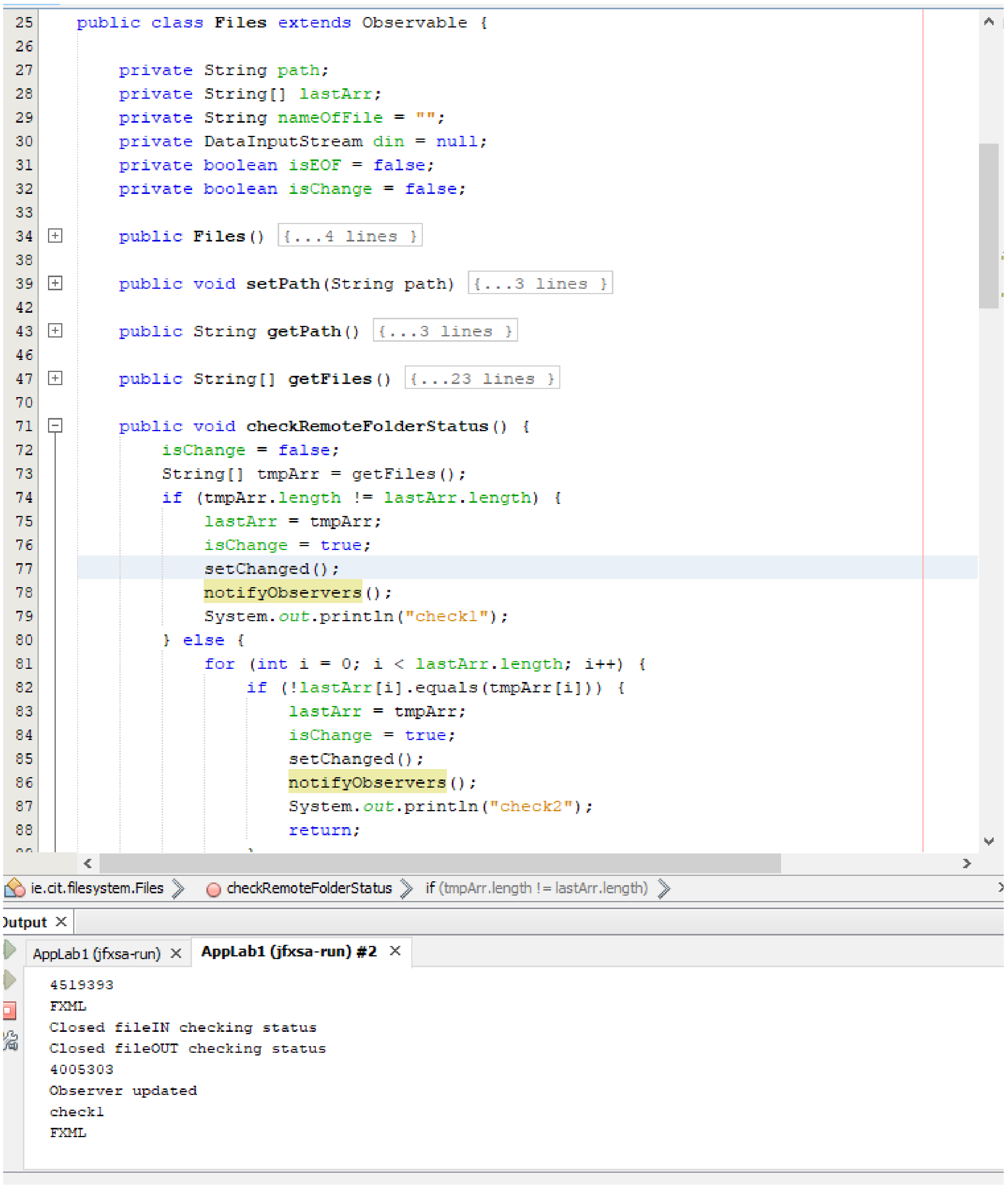
Also, the remote folder uses a **Singleton** and **Observer pattern**.



```
public class RemoteFolder implements FolderMonitor, Observer {

    private static RemoteFolder INSTANCE;
    private static Files files;

    private RemoteFolder() {}

    public static RemoteFolder getInstance(String path) {
        if (INSTANCE == null) {
            INSTANCE = new RemoteFolder();
            files = new Files(); //Observable
            files.setPath(path);
            files.addObserver(INSTANCE); //adding observer
        }
        return INSTANCE;
    }

    @Override
    public void update(Observable observable, Object arg) {
        System.out.println("Observer updated");
        files = (Files) observable;
    }
```

The class File implements Observer. When any change happens to the remote folder such; add, remove, even rename, the observer is notified.

In the console, we can see that Observer was updated because was notified by “check1”: the file was added or removed.

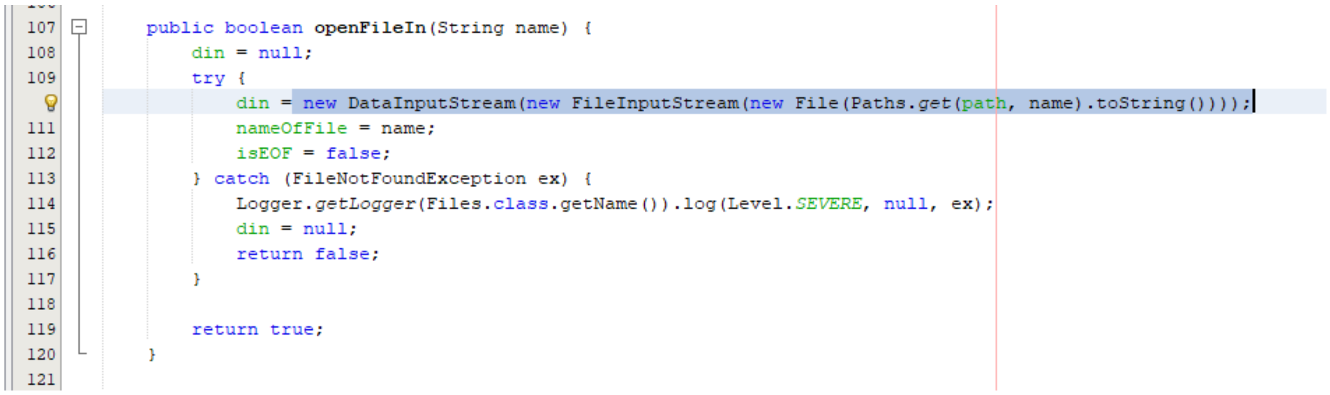


```java
public class Files extends Observable {

    private String path;
    private String[] lastArr;
    private String nameOfFile = "";
    private DataInputStream din = null;
    private boolean isEOF = false;
    private boolean isChange = false;

    public Files() {...4 lines }

    public void setPath(String path) {...3 lines }

    public String getPath() {...3 lines }

    public String[] getFiles() {...23 lines }

    public void checkRemoteFolderStatus() {
        isChange = false;
        String[] tmpArr = getFiles();
        if (tmpArr.length != lastArr.length) {
            lastArr = tmpArr;
            isChange = true;
            setChanged();
            notifyObservers();
            System.out.println("check1");
        } else {
            for (int i = 0; i < lastArr.length; i++) {
                if (!lastArr[i].equals(tmpArr[i])) {
                    lastArr = tmpArr;
                    isChange = true;
                    setChanged();
                    notifyObservers();
                    System.out.println("check2");
                    return;
```

```
4519393
FXML
Closed fileIN checking status
Closed fileOUT checking status
4005303
Observer updated
check1
FXML
```

## Decorative Pattern

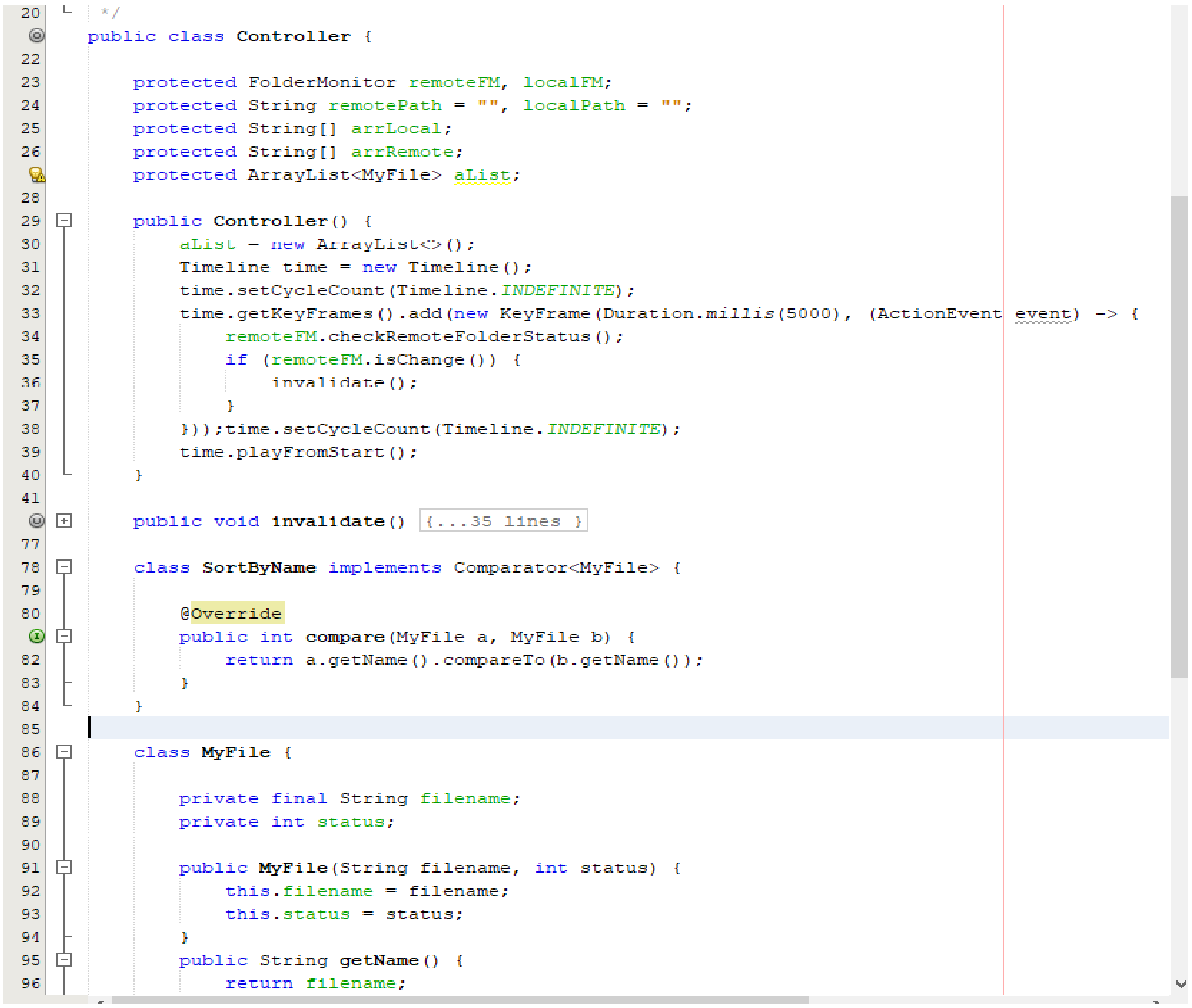
Example of a decorative pattern. This is used in *openFileIn* and on *openFileOut* method.



```java
    public boolean openFileIn(String name) {
        din = null;
        try {
            din = new DataInputStream(new FileInputStream(new File(Paths.get(path, name).toString())));
            nameOfFile = name;
            isEOF = false;
        } catch (FileNotFoundException ex) {
            Logger.getLogger(Files.class.getName()).log(Level.SEVERE, null, ex);
            din = null;
            return false;
        }

        return true;
    }

```

## Facade Pattern

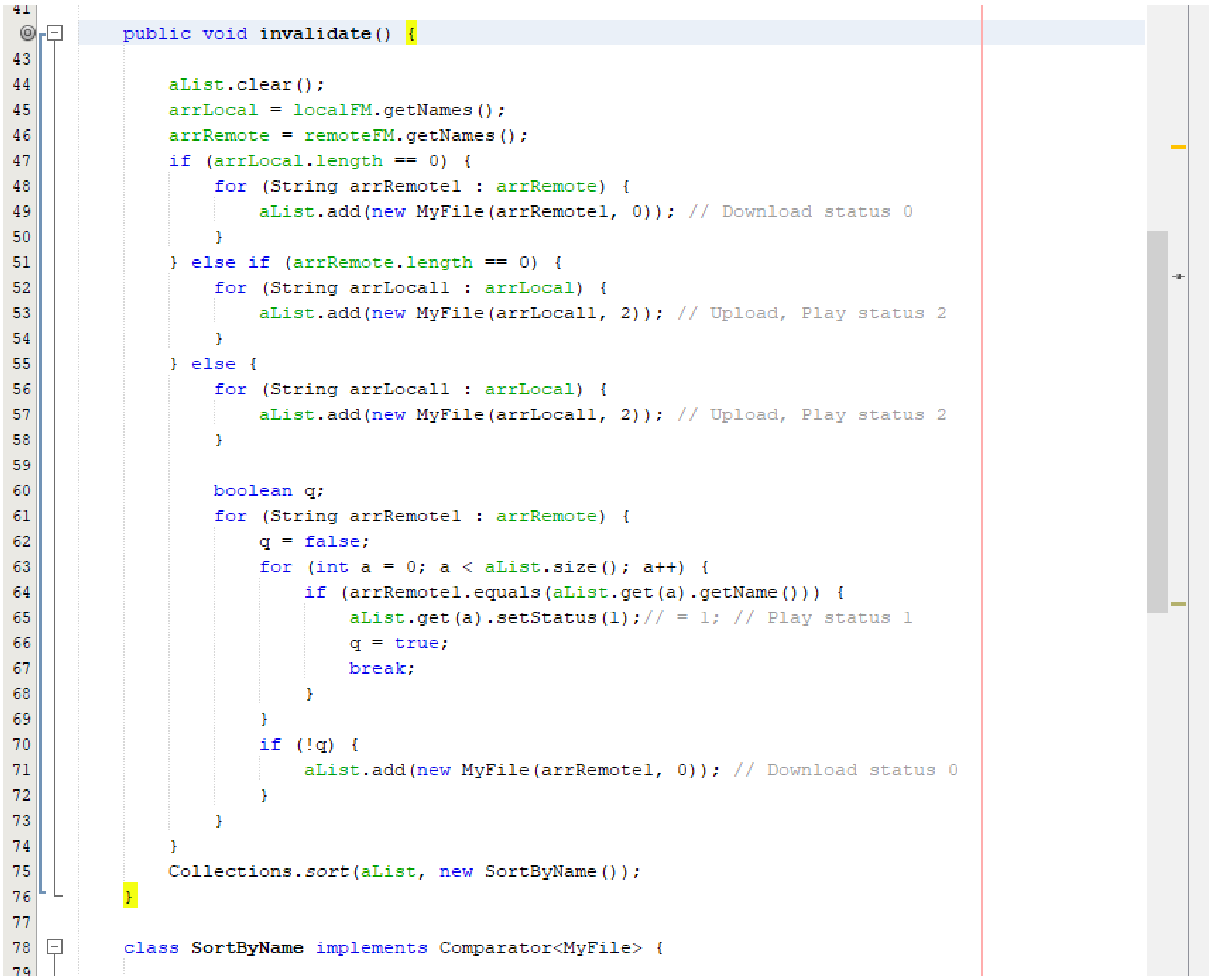
Multiple classes are used in one class for simplification of the complex.



```java
 */
public class Controller {

    protected FolderMonitor remoteFM, localFM;
    protected String remotePath = "", localPath = "";
    protected String[] arrLocal;
    protected String[] arrRemote;
    protected ArrayList<MyFile> aList;

    public Controller() {
        aList = new ArrayList<>();
        Timeline time = new Timeline();
        time.setCycleCount(Timeline.INDEFINITE);
        time.getKeyFrames().add(new KeyFrame(Duration.millis(5000), (ActionEvent event) -> {
            remoteFM.checkRemoteFolderStatus();
            if (remoteFM.isChange()) {
                invalidate();
            }
        }));time.setCycleCount(Timeline.INDEFINITE);
        time.playFromStart();
    }

    public void invalidate() {...35 lines }

    class SortByName implements Comparator<MyFile> {

        @Override
        public int compare(MyFile a, MyFile b) {
            return a.getName().compareTo(b.getName());
        }
    }

    class MyFile {

        private final String filename;
        private int status;

        public MyFile(String filename, int status) {
            this.filename = filename;
            this.status = status;
        }
        public String getName() {
            return filename;
```

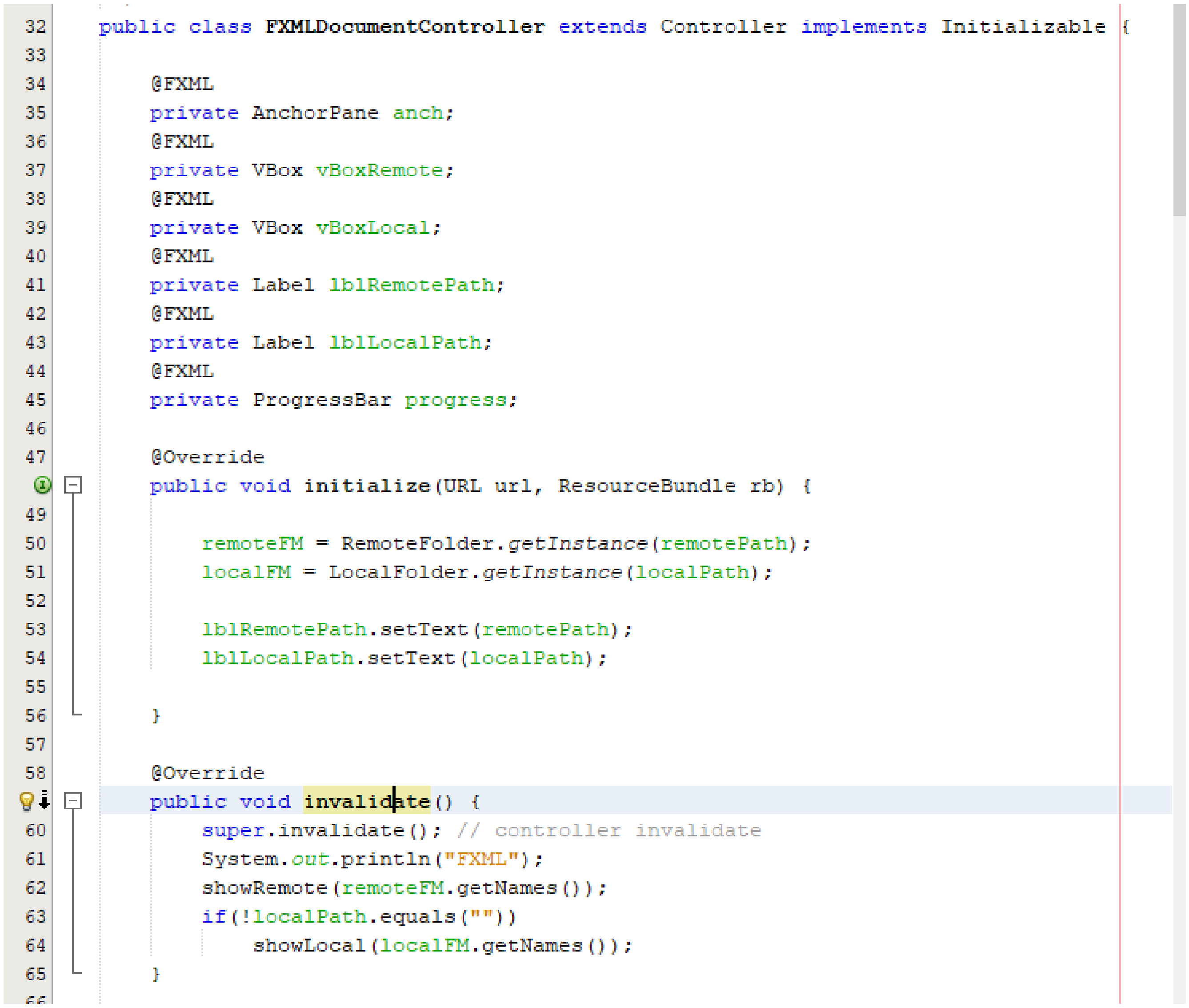
## Separating logic from view

The method *invalidate* is making from two arrays one *ArrayList*.
In the *FXMLDocumentController* the overridden *invalidate* method displaying buttons and labels.



```java
    public void invalidate() {

        aList.clear();
        arrLocal = localFM.getNames();
        arrRemote = remoteFM.getNames();
        if (arrLocal.length == 0) {
            for (String arrRemotel : arrRemote) {
                aList.add(new MyFile(arrRemotel, 0)); // Download status 0
            }
        } else if (arrRemote.length == 0) {
            for (String arrLocall : arrLocal) {
                aList.add(new MyFile(arrLocall, 2)); // Upload, Play status 2
            }
        } else {
            for (String arrLocall : arrLocal) {
                aList.add(new MyFile(arrLocall, 2)); // Upload, Play status 2
            }

            boolean q;
            for (String arrRemotel : arrRemote) {
                q = false;
                for (int a = 0; a < aList.size(); a++) {
                    if (arrRemotel.equals(aList.get(a).getName())) {
                        aList.get(a).setStatus(1);// = 1; // Play status 1
                        q = true;
                        break;
                    }
                }
                if (!q) {
                    aList.add(new MyFile(arrRemotel, 0)); // Download status 0
                }
            }
        }
        Collections.sort(aList, new SortByName());
    }

    class SortByName implements Comparator<MyFile> {
```

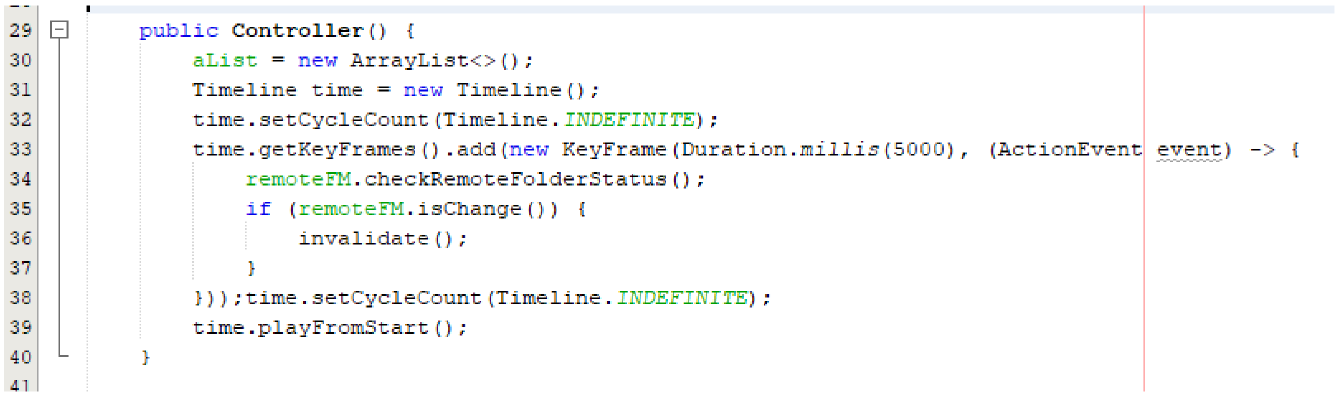
At line 32 is inheritance used. At line 60 is invalidate method called from Controller by super.



```
32    public class FXMLDocumentController extends Controller implements Initializable {
33
34        @FXML
35        private AnchorPane anch;
36        @FXML
37        private VBox vBoxRemote;
38        @FXML
39        private VBox vBoxLocal;
40        @FXML
41        private Label lblRemotePath;
42        @FXML
43        private Label lblLocalPath;
44        @FXML
45        private ProgressBar progress;
46
47        @Override
          public void initialize(URL url, ResourceBundle rb) {
49
50            remoteFM = RemoteFolder.getInstance(remotePath);
51            localFM = LocalFolder.getInstance(localPath);
52
53            lblRemotePath.setText(remotePath);
54            lblLocalPath.setText(localPath);
55
56        }
57
58        @Override
          public void invalidate() {
60            super.invalidate(); // controller invalidate
61            System.out.println("FXML");
62            showRemote(remoteFM.getNames());
63            if(!localPath.equals(""))
64                showLocal(localFM.getNames());
65        }
```

## Threading

### Timeline

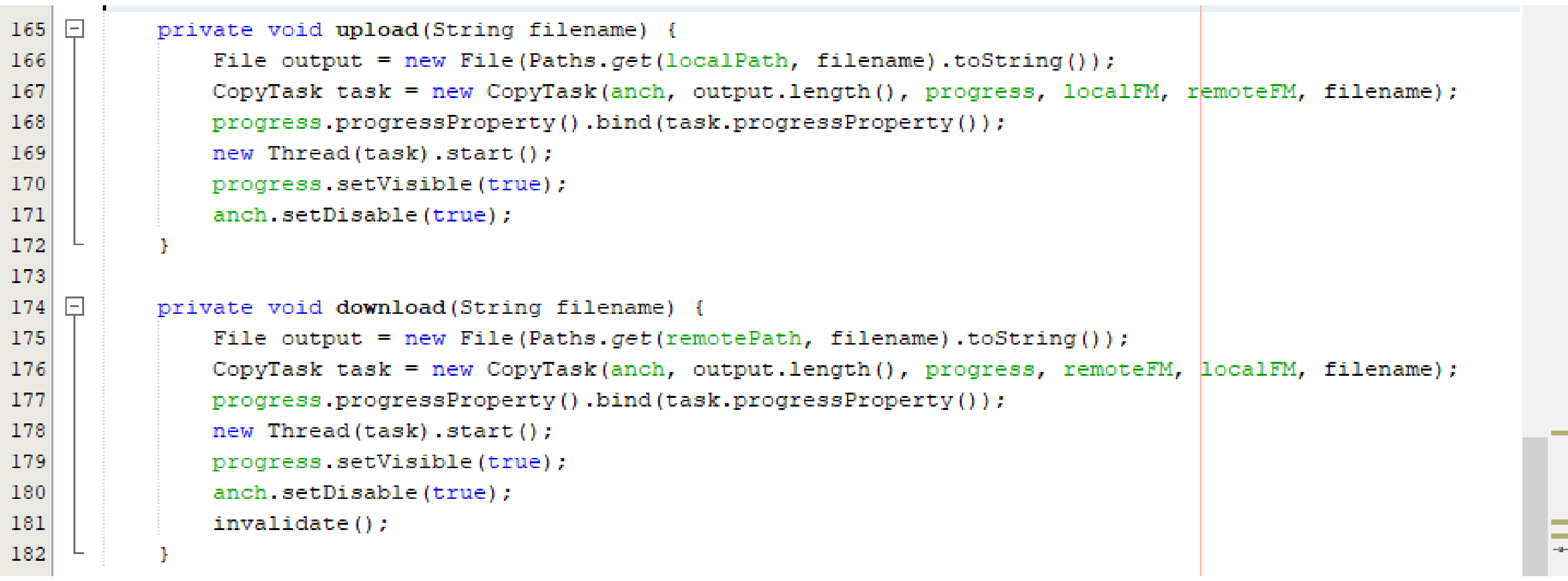


```
public Controller() {
    aList = new ArrayList<>();
    Timeline time = new Timeline();
    time.setCycleCount(Timeline.INDEFINITE);
    time.getKeyFrames().add(new KeyFrame(Duration.millis(5000), (ActionEvent event) -> {
        remoteFM.checkRemoteFolderStatus();
        if (remoteFM.isChange()) {
            invalidate();
        }
    }));time.setCycleCount(Timeline.INDEFINITE);
    time.playFromStart();
}
```

In the constructor of the *Controller*, we are checking for *isChange* in the remote folder every 5 seconds. If *Observer* notifies a change, the *invalidate* method in *Controller* prepare the *ArrayList* and *invalidate* in *FXMLDocumentController* displays the changes.

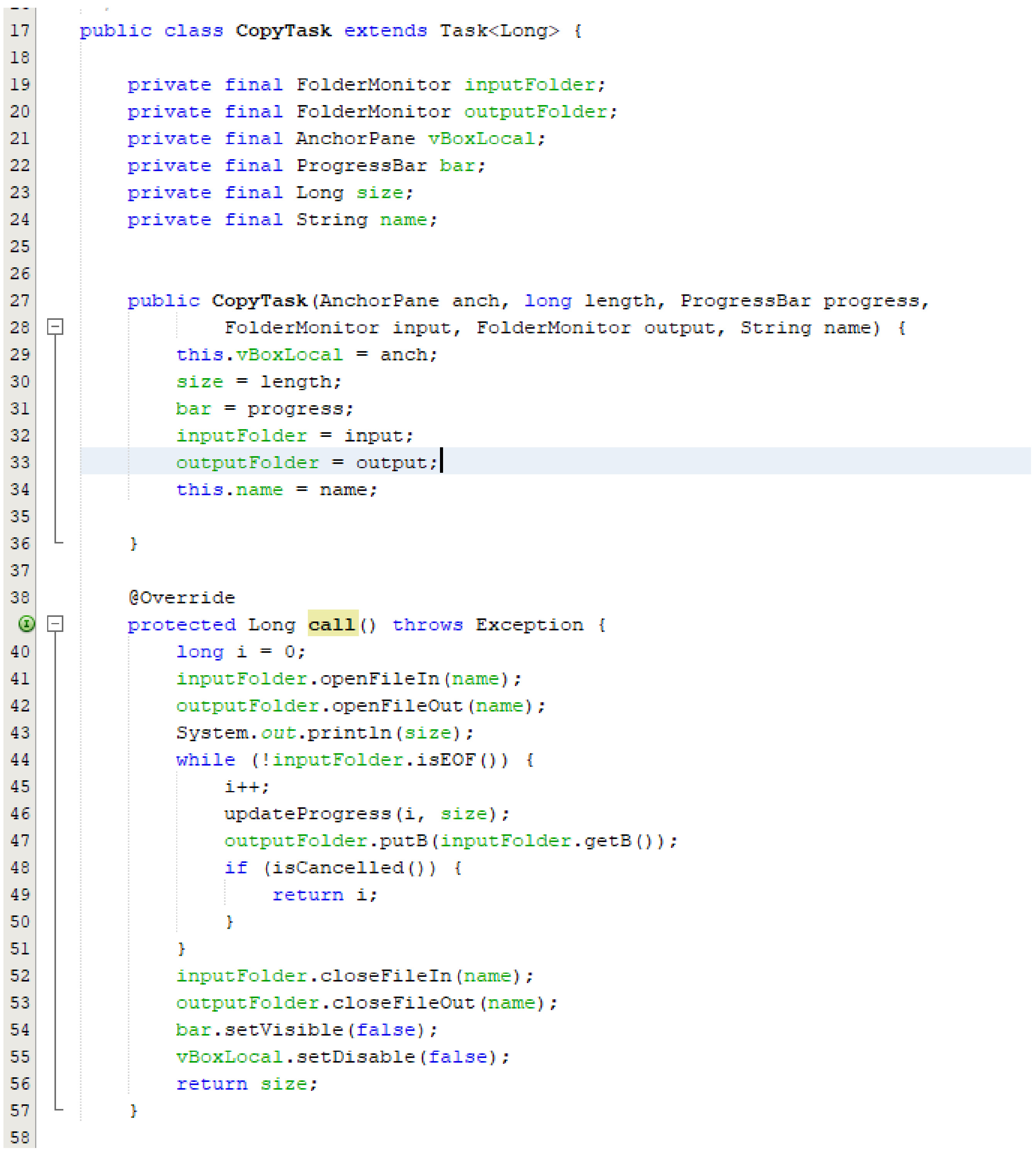
## Task

This type of thread is used for download or upload, the CopyTask constructor is filled and Thread is started.



```java
    private void upload(String filename) {
        File output = new File(Paths.get(localPath, filename).toString());
        CopyTask task = new CopyTask(anch, output.length(), progress, localFM, remoteFM, filename);
        progress.progressProperty().bind(task.progressProperty());
        new Thread(task).start();
        progress.setVisible(true);
        anch.setDisable(true);
    }

    private void download(String filename) {
        File output = new File(Paths.get(remotePath, filename).toString());
        CopyTask task = new CopyTask(anch, output.length(), progress, remoteFM, localFM, filename);
        progress.progressProperty().bind(task.progressProperty());
        new Thread(task).start();
        progress.setVisible(true);
        anch.setDisable(true);
        invalidate();
    }
```

The size of a file is passed to the task for the displaying progress bar.
In the call method, we are doing Byte by Byte copy and updating progress into the progress bar.
When a copy is a finish, we hide a progress bar (just for an aesthetic reason) and enabling all buttons.

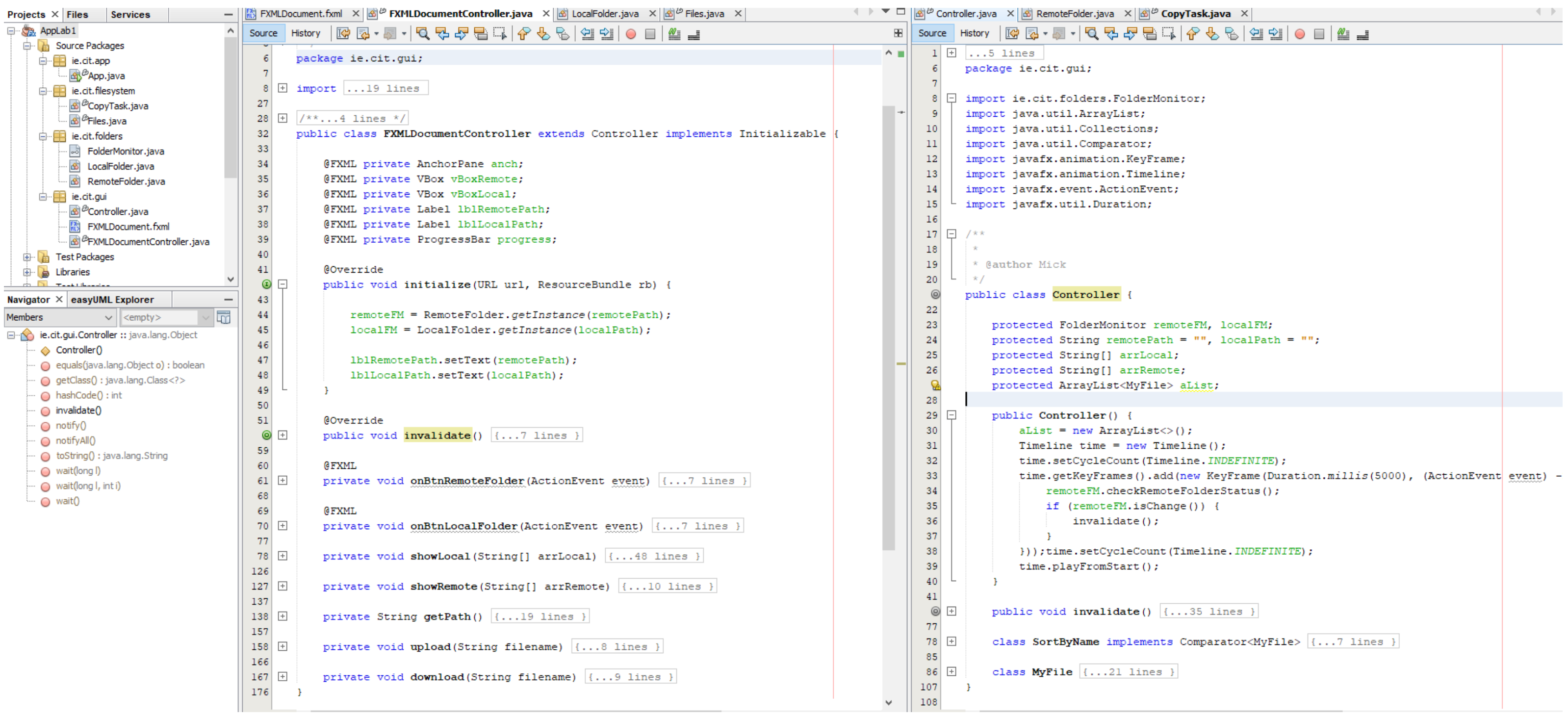


```java
public class CopyTask extends Task<Long> {

    private final FolderMonitor inputFolder;
    private final FolderMonitor outputFolder;
    private final AnchorPane vBoxLocal;
    private final ProgressBar bar;
    private final Long size;
    private final String name;


    public CopyTask(AnchorPane anch, long length, ProgressBar progress,
            FolderMonitor input, FolderMonitor output, String name) {
        this.vBoxLocal = anch;
        size = length;
        bar = progress;
        inputFolder = input;
        outputFolder = output;
        this.name = name;

    }

    @Override
    protected Long call() throws Exception {
        long i = 0;
        inputFolder.openFileIn(name);
        outputFolder.openFileOut(name);
        System.out.println(size);
        while (!inputFolder.isEOF()) {
            i++;
            updateProgress(i, size);
            outputFolder.putB(inputFolder.getB());
            if (isCancelled()) {
                return i;
            }
        }
        inputFolder.closeFileIn(name);
        outputFolder.closeFileOut(name);
        bar.setVisible(false);
        vBoxLocal.setDisable(false);
        return size;
    }
```

*See call method.*

# Architecture

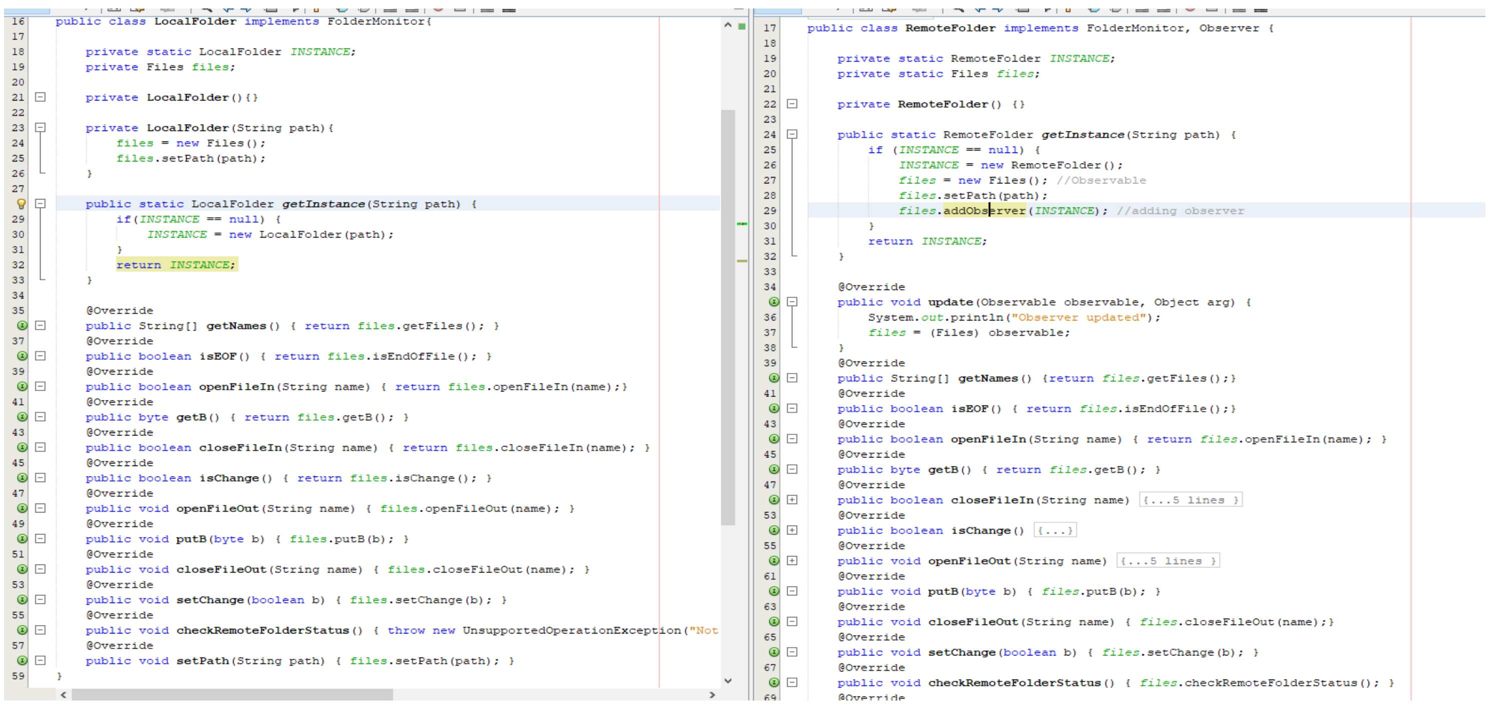
## GUI package



```java
package ie.cit.gui;

import ...19 lines

/**...4 lines */
public class FXMLDocumentController extends Controller implements Initializable {

    @FXML private AnchorPane anch;
    @FXML private VBox vBoxRemote;
    @FXML private VBox vBoxLocal;
    @FXML private Label lblRemotePath;
    @FXML private Label lblLocalPath;
    @FXML private ProgressBar progress;

    @Override
    public void initialize(URL url, ResourceBundle rb) {

        remoteFM = RemoteFolder.getInstance(remotePath);
        localFM = LocalFolder.getInstance(localPath);

        lblRemotePath.setText(remotePath);
        lblLocalPath.setText(localPath);
    }

    @Override
    public void invalidate() {...7 lines }

    @FXML
    private void onBtnRemoteFolder(ActionEvent event) {...7 lines }

    @FXML
    private void onBtnLocalFolder(ActionEvent event) {...7 lines }

    private void showLocal(String[] arrLocal) {...48 lines }

    private void showRemote(String[] arrRemote) {...10 lines }

    private String getPath() {...19 lines }

    private void upload(String filename) {...8 lines }

    private void download(String filename) {...9 lines }
}
```

```java
...5 lines
package ie.cit.gui;

import ie.cit.folders.FolderMonitor;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.event.ActionEvent;
import javafx.util.Duration;

/**
 *
 * @author Mick
 */
public class Controller {

    protected FolderMonitor remoteFM, localFM;
    protected String remotePath = "", localPath = "";
    protected String[] arrLocal;
    protected String[] arrRemote;
    protected ArrayList<MyFile> aList;

    public Controller() {
        aList = new ArrayList<>();
        Timeline time = new Timeline();
        time.setCycleCount(Timeline.INDEFINITE);
        time.getKeyFrames().add(new KeyFrame(Duration.millis(5000), (ActionEvent event) -
            remoteFM.checkRemoteFolderStatus();
            if (remoteFM.isChange()) {
                invalidate();
            }
        }));time.setCycleCount(Timeline.INDEFINITE);
        time.playFromStart();
    }

    public void invalidate() {...35 lines }

    class SortByName implements Comparator<MyFile> {...7 lines }

    class MyFile {...21 lines }
}
```

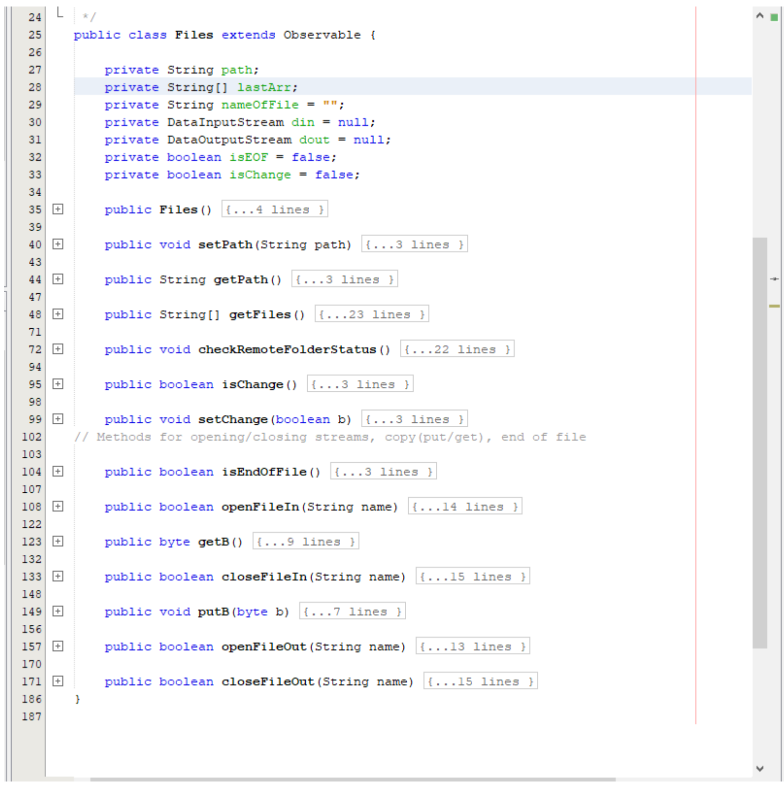
## Folders package



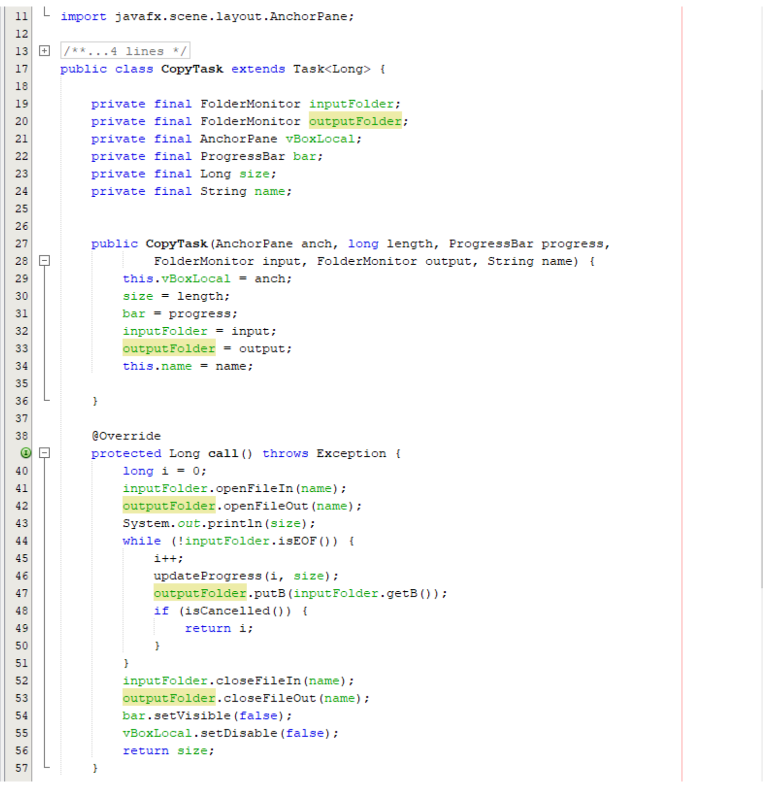
```java
public class LocalFolder implements FolderMonitor{

    private static LocalFolder INSTANCE;
    private Files files;

    private LocalFolder(){}

    private LocalFolder(String path){
        files = new Files();
        files.setPath(path);
    }

    public static LocalFolder getInstance(String path) {
        if(INSTANCE == null) {
            INSTANCE = new LocalFolder(path);
        }
        return INSTANCE;
    }

    @Override
    public String[] getNames() { return files.getFiles(); }
    @Override
    public boolean isEOF() { return files.isEndOfFile(); }
    @Override
    public boolean openFileIn(String name) { return files.openFileIn(name);}
    @Override
    public byte getB() { return files.getB(); }
    @Override
    public boolean closeFileIn(String name) { return files.closeFileIn(name); }
    @Override
    public boolean isChange() { return files.isChange(); }
    @Override
    public void openFileOut(String name) { files.openFileOut(name); }
    @Override
    public void putB(byte b) { files.putB(b); }
    @Override
    public void closeFileOut(String name) { files.closeFileOut(name); }
    @Override
    public void setChange(boolean b) { files.setChange(b); }
    @Override
    public void checkRemoteFolderStatus() { throw new UnsupportedOperationException("Not
    @Override
    public void setPath(String path) { files.setPath(path); }
}
```

```java
public class RemoteFolder implements FolderMonitor, Observer {

    private static RemoteFolder INSTANCE;
    private static Files files;

    private RemoteFolder() {}

    public static RemoteFolder getInstance(String path) {
        if (INSTANCE == null) {
            INSTANCE = new RemoteFolder();
            files = new Files(); //Observable
            files.setPath(path);
            files.addObserver(INSTANCE); //adding observer
        }
        return INSTANCE;
    }

    @Override
    public void update(Observable observable, Object arg) {
        System.out.println("Observer updated");
        files = (Files) observable;
    }
    @Override
    public String[] getNames() {return files.getFiles();}
    @Override
    public boolean isEOF() { return files.isEndOfFile();}
    @Override
    public boolean openFileIn(String name) { return files.openFileIn(name); }
    @Override
    public byte getB() { return files.getB(); }
    @Override
    public boolean closeFileIn(String name) {...5 lines }
    @Override
    public boolean isChange() {...}
    @Override
    public void openFileOut(String name) {...5 lines }
    @Override
    public void putB(byte b) { files.putB(b); }
    @Override
    public void closeFileOut(String name) { files.closeFileOut(name);}
    @Override
    public void setChange(boolean b) { files.setChange(b); }
    @Override
    public void checkRemoteFolderStatus() { files.checkRemoteFolderStatus(); }
    @Override
```

## Filesystem packages

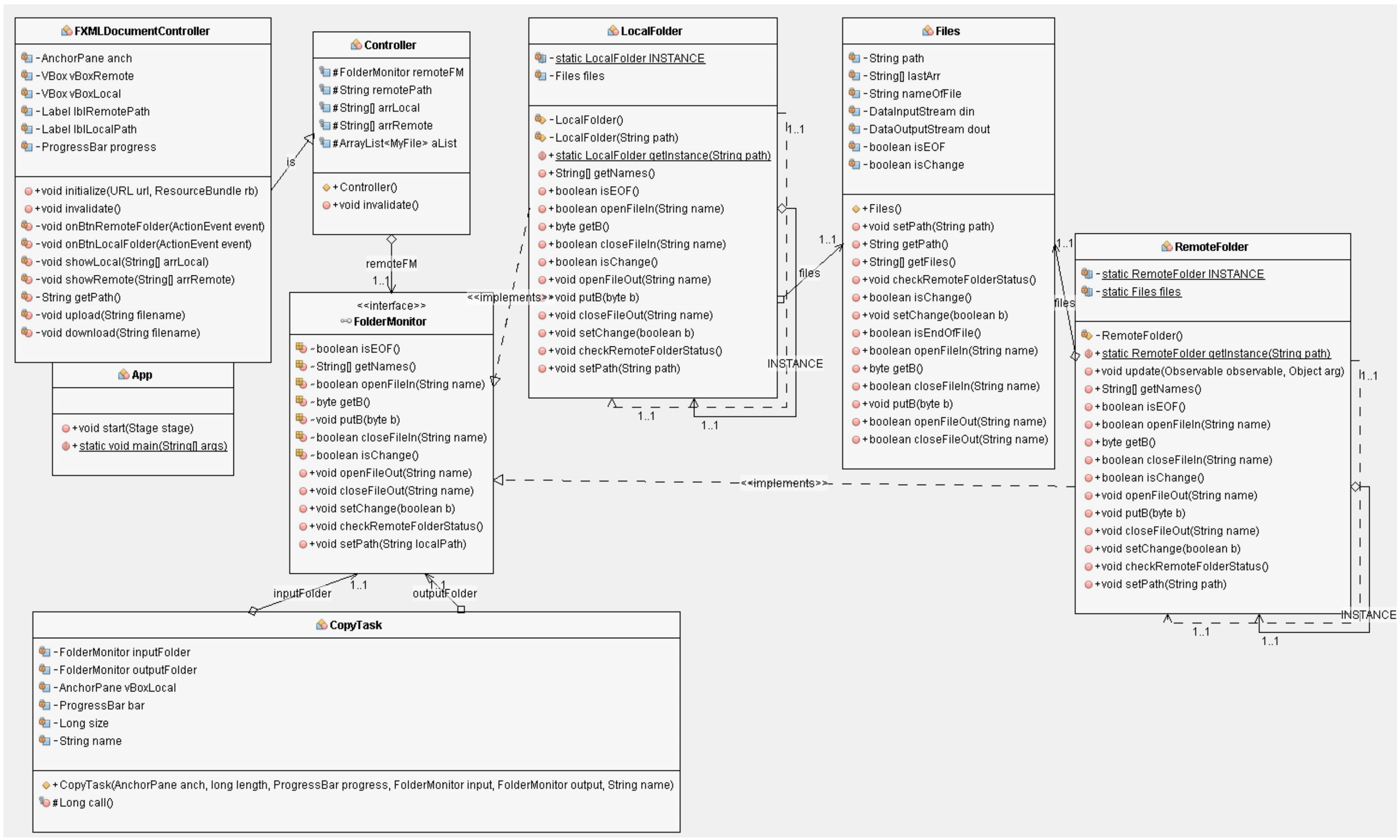


```java
 */
public class Files extends Observable {

    private String path;
    private String[] lastArr;
    private String nameOfFile = "";
    private DataInputStream din = null;
    private DataOutputStream dout = null;
    private boolean isEOF = false;
    private boolean isChange = false;

    public Files() {...4 lines }

    public void setPath(String path) {...3 lines }

    public String getPath() {...3 lines }

    public String[] getFiles() {...23 lines }

    public void checkRemoteFolderStatus() {...22 lines }

    public boolean isChange() {...3 lines }

    public void setChange(boolean b) {...3 lines }
// Methods for opening/closing streams, copy(put/get), end of file

    public boolean isEndOfFile() {...3 lines }

    public boolean openFileIn(String name) {...14 lines }

    public byte getB() {...9 lines }

    public boolean closeFileIn(String name) {...15 lines }

    public void putB(byte b) {...7 lines }

    public boolean openFileOut(String name) {...13 lines }

    public boolean closeFileOut(String name) {...15 lines }
}
```



```java
import javafx.scene.layout.AnchorPane;

/**...4 lines */
public class CopyTask extends Task<Long> {

    private final FolderMonitor inputFolder;
    private final FolderMonitor outputFolder;
    private final AnchorPane vBoxLocal;
    private final ProgressBar bar;
    private final Long size;
    private final String name;


    public CopyTask(AnchorPane anch, long length, ProgressBar progress,
            FolderMonitor input, FolderMonitor output, String name) {
        this.vBoxLocal = anch;
        size = length;
        bar = progress;
        inputFolder = input;
        outputFolder = output;
        this.name = name;

    }

    @Override
    protected Long call() throws Exception {
        long i = 0;
        inputFolder.openFileIn(name);
        outputFolder.openFileOut(name);
        System.out.println(size);
        while (!inputFolder.isEOF()) {
            i++;
            updateProgress(i, size);
            outputFolder.putB(inputFolder.getB());
            if (isCancelled()) {
                return i;
            }
        }
        inputFolder.closeFileIn(name);
        outputFolder.closeFileOut(name);
        bar.setVisible(false);
        vBoxLocal.setDisable(false);
        return size;
    }
```

## UML diagram


FXMLDocumentController
-AnchorPane anch
-VBox vBoxRemote
-VBox vBoxLocal
-Label lblRemotePath
-Label lblLocalPath
-ProgressBar progress
+void initialize(URL url, ResourceBundle rb)
+void invalidate()
-void onBtnRemoteFolder(ActionEvent event)
-void onBtnLocalFolder(ActionEvent event)
-void showLocal(String[] arrLocal)
-void showRemote(String[] arrRemote)
-String getPath()
-void upload(String filename)
-void download(String filename)
is
Controller
#FolderMonitor remoteFM
#String remotePath
#String[] arrLocal
#String[] arrRemote
#ArrayList<MyFile> aList
+Controller()
+void invalidate()
remoteFM
1..1
App
+void start(Stage stage)
+static void main(String[] args)
<<interface>>
FolderMonitor
~boolean isEOF()
~String[] getNames()
~boolean openFileIn(String name)
~byte getB()
~void putB(byte b)
~boolean closeFileIn(String name)
~boolean isChange()
+void openFileOut(String name)
+void closeFileOut(String name)
+void setChange(boolean b)
+void checkRemoteFolderStatus()
+void setPath(String localPath)
<<implements>>
LocalFolder
-static LocalFolder INSTANCE
-Files files
-LocalFolder()
-LocalFolder(String path)
+static LocalFolder getInstance(String path)
+String[] getNames()
+boolean isEOF()
+boolean openFileIn(String name)
+byte getB()
+boolean closeFileIn(String name)
+boolean isChange()
+void openFileOut(String name)
+void putB(byte b)
+void closeFileOut(String name)
+void setChange(boolean b)
+void checkRemoteFolderStatus()
+void setPath(String path)
1..1
INSTANCE
1..1
1..1
files
1..1
Files
-String path
-String[] lastArr
-String nameOfFile
-DataInputStream din
-DataOutputStream dout
-boolean isEOF
-boolean isChange
+Files()
+void setPath(String path)
+String getPath()
+String[] getFiles()
+void checkRemoteFolderStatus()
+boolean isChange()
+void setChange(boolean b)
+boolean isEndOfFile()
+boolean openFileIn(String name)
+byte getB()
+boolean closeFileIn(String name)
+void putB(byte b)
+boolean openFileOut(String name)
+boolean closeFileOut(String name)
1..1
files
RemoteFolder
-static RemoteFolder INSTANCE
-static Files files
-RemoteFolder()
+static RemoteFolder getInstance(String path)
+void update(Observable observable, Object arg)
+String[] getNames()
+boolean isEOF()
+boolean openFileIn(String name)
+byte getB()
+boolean closeFileIn(String name)
+boolean isChange()
+void openFileOut(String name)
+void putB(byte b)
+void closeFileOut(String name)
+void setChange(boolean b)
+void checkRemoteFolderStatus()
+void setPath(String path)
1..1
INSTANCE
1..1
1..1
<<implements>>
inputFolder
1..1
1..1
outputFolder
CopyTask
-FolderMonitor inputFolder
-FolderMonitor outputFolder
-AnchorPane vBoxLocal
-ProgressBar bar
-Long size
-String name
+CopyTask(AnchorPane anch, long length, ProgressBar progress, FolderMonitor input, FolderMonitor output, String name)
#Long call()

## Conclusion

The application is implemented in a few lines of code because of the structure. For sensible packages is easy to extend, rebuild or move, thankfully to interfaces, packages and patterns used.