|  |  |
| --- | --- |
| H:\LOGO-NXV\Hai__090908__02_1_den.jpg | Faculty of Information Technology  H A N O I U N I V E R S I T Y |

61FIT3NPR – Network Programming

Java Streams

1. **Read from an URL and write the result to a file on the disk:**

We use 2 classes: Downloader and DownloadMain

**package** tut3;

**import** java.io.\*;

**import** java.net.\*;

**public** **class** Downloader {

**private** URL url;

// Constructs downloader to read from the given URL.

**public** Downloader(String urlString) **throws** MalformedURLException {

url = **new** URL(urlString);

}

// Reads downloader's URL and writes contents to the given file.

**public** **void** download(String targetFileName) **throws** IOException {

InputStream in = url.openStream();

FileOutputStream out = **new** FileOutputStream(targetFileName);

**while** (**true**) {

**int** n = in.read();

**if** (n == -1) { // -1 means end-of-file

**break**;

}

out.write(n);

}

in.close();

out.close();

}

}

**package** tut3;

**import** java.io.\*;

**import** java.net.\*;

**import** java.util.\*;

**public** **class** DownloadMain {

**public** **static** **void** main(String[] args) {

Scanner console = **new** Scanner(System.***in***);

System.***out***.print("URL to download? ");

String urlString = console.nextLine();

Downloader down = **null**; // create a downloader;

**while** (down == **null**) { // re-prompt the user if this fails

**try** {

down = **new** Downloader(urlString);

} **catch** (MalformedURLException e) {

System.***out***.print("Bad URL! Try again: ");

urlString = console.nextLine();

}

}

System.***out***.print("Target file name: ");

String targetFileName = console.nextLine();

**try** { // download bytes to file (print error if it fails)

down.download(targetFileName);

System.***out***.println("Downloaded successfully") ;

} **catch** (IOException e) {

System.***out***.println("I/O error: " + e.getMessage());

}

}

}

1. **Write class TallyDownloader to add behavior to Downloader**

public TallyDownloader(String url)

public void download(String targetFileName)

(Downloads the file, and also prints the file to the console, and prints the number of occurrences of each kind of character in the file.)

**package** tut3;

**import** java.io.\*;

**import** java.net.\*;

**import** java.util.\*;

**public** **class** TallyDownloader **extends** Downloader {

**public** TallyDownloader(String url) **throws** MalformedURLException {

**super**(url); // call Downloader constructor

}

// Reads from URL and prints file contents and tally of each char.

**public** **void** download(String targetFileName) **throws** IOException {

**super**.download(targetFileName);

Map<Character, Integer> counts = **new** TreeMap<Character, Integer>();

FileInputStream in = **new** FileInputStream(targetFileName);

**while** (**true**) {

**int** n = in.read();

**if** (n == -1) {

**break**;

}

**char** ch = (**char**) n;

**if** (counts.containsKey(ch)) {

counts.put(ch, counts.get(ch) + 1);

} **else** {

counts.put(ch, 1);

}

System.***out***.print(ch);

}

in.close();

System.***out***.println(counts); // print map of char -> int

}

}

**package** tut3;

**import** java.io.\*;

**import** java.net.\*;

**import** java.util.\*;

**public** **class** DownloadMain {

**public** **static** **void** main(String[] args) {

Scanner console = **new** Scanner(System.***in***);

System.***out***.print("URL to download? ");

String urlString = console.nextLine();

Downloader down = **null**; // create a tallying downloader;

**while** (down == **null**) { // re-prompt the user if this fails

**try** {

down = **new** TallyDownloader(urlString);

} **catch** (MalformedURLException e) {

System.***out***.print("Bad URL! Try again: ");

urlString = console.nextLine();

}

}

System.***out***.print("Target file name: ");

String targetFileName = console.nextLine();

**try** { // download bytes to file (print error if it fails)

down.download(targetFileName);

System.***out***.println("Downloaded successfully") ;

} **catch** (IOException e) {

System.***out***.println("I/O error: " + e.getMessage());

}

}

}

1. **An example how to read from a file and print out the content to the console**

**package** tut3.javaio.stream;

**import** java.io.FileInputStream;

**import** java.io.InputStream;

**public** **class** InputStreamExample2 {

**public** **static** **void** main(String[] args) {

**try** {

InputStream in = **new** FileInputStream("data.txt");

**byte**[] temp = **new** **byte**[10];

**int** i = -1;

**while** ((i = in.read(temp)) != -1) {

String s = **new** String(temp, 0, i);

System.***out***.println(s);

}

in.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

**An example how to write characters to a file:**

**package** tut3.javaio.stream;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.OutputStream;

**public** **class** OutputStreamExample2 {

**public** **static** **void** main(String[] args) {

**try** {

File dir = **new** File("./Test");

dir.mkdirs();

OutputStream os = **new** FileOutputStream("./Test/test\_writerOutputStream.txt");

**byte[] by = new byte[] { 'H', 'e', 'l', 'l', 'o', ' ', 31, 34, 92, 10 };** //10 = "\n" new line , 31 = control code chart , 34 = """ , 92 = "\"

**byte**[] by2 = **new** **byte**[] { 'H', 'e', 'l', 'l', 'o', ' ', 'b', 'o',

'y' };

os.write(by);

os.flush();

os.write(by2);

os.close();

} **catch** (Exception e) {

e.printStackTrace();

}

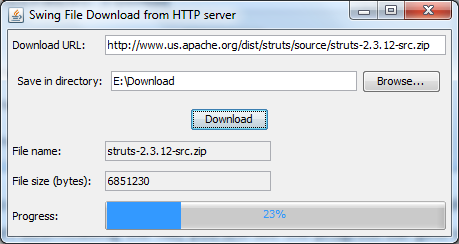
}

}

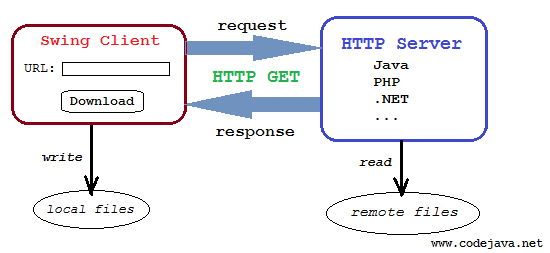
Another: <https://www.codejava.net/java-se/networking/use-httpurlconnection-to-download-file-from-an-http-url>

1. **Java Swing application to download files from HTTP server with progress bar**

This tutorial is about how to build a Swing application that downloads a file from a HTTP server through a URL. The download application would look like this:



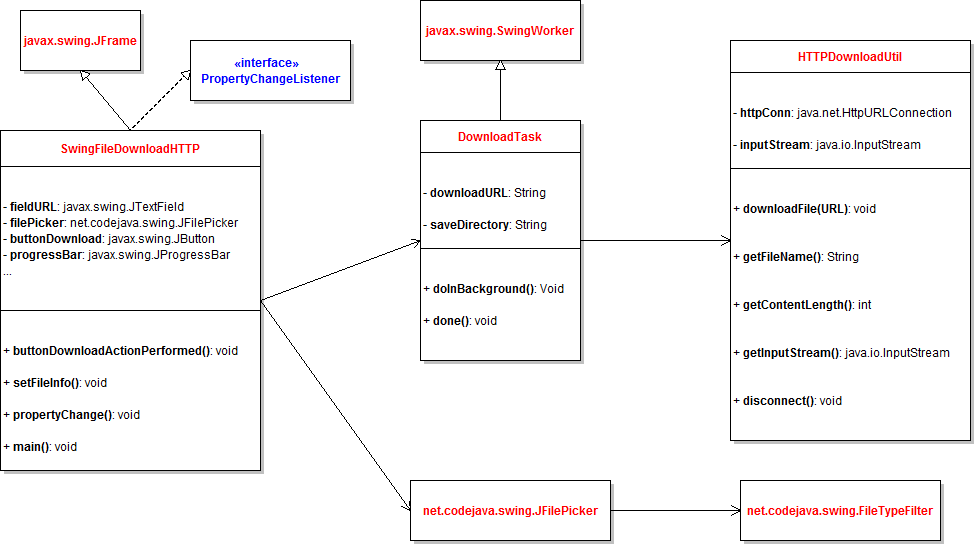
Its workflow is explained in this diagram:



The principal solution for this download application involves in using the java.net.HttpURLConnection class to initiates an HTTP connection with the server. The Swing client will parse HTTP headers sent from the server to determine information about the file to be downloaded such as file type, filename and content length. Then opening the HTTP connection’s input stream to read bytes transferred from the server. And an output stream is created on the client side to save the file on disk.

For more information about using HttpURLConnection class to download file, see the tutorial: [Use HttpURLConnection to download file from an HTTP URL](https://www.codejava.net/java-se/networking/use-httpurlconnection-to-download-file-from-an-http-url).

The following class diagram illustrates how the application will be designed:



As we can see, the application consists of the following classes:

* + HTTPDownloadUtil: encapsulates the core functionality of the application - download file from an HTTP URL. This class is implemented based on the tutorial [Use HttpURLConnection to download file from an HTTP URL](https://www.codejava.net/java-se/networking/use-httpurlconnection-to-download-file-from-an-http-url), but is modified to work with updating download progress while the file is being transferred.
  + DownloadTask: executes the file download in a background thread rather in the Swing’s event dispatcher thread (EDT), so the GUI will not become unresponsive when the download is taking place.
  + SwingFileDownloadHTTP: is the main application which displays the GUI that allows users to enter a download URL, choose a location to save the file, and start downloading.
  + JFilePicker and FileTypeFilter: These components are used by the SwingFileDownloadHTTP class to show a file chooser. These classes are taken from the article [File picker component in Swing](https://www.codejava.net/java-se/swing/file-picker-component-in-swing).

Let’s see how these classes are implemented in details.

## 1. Code of the HTTPDownloadUtil class

**package** net.codejava.swing.download;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.net.HttpURLConnection;

**import** java.net.URL;

/\*\*

 \* A utility that downloads a file from a URL.

 \*

 \* @author www.codejava.net

 \*

 \*/

**public** **class** HTTPDownloadUtil {

**private** HttpURLConnection httpConn;

    /\*\*

     \* hold input stream of HttpURLConnection

     \*/

**private** InputStream inputStream;

**private** String fileName;

**private** **int** contentLength;

    /\*\*

     \* Downloads a file from a URL

     \*

     \* @param fileURL

     \*            HTTP URL of the file to be downloaded

     \* @throws IOException

     \*/

**public** **void** downloadFile(String fileURL) **throws** IOException {

        URL url = **new** URL(fileURL);

        httpConn = (HttpURLConnection) url.openConnection();

**int** responseCode = httpConn.getResponseCode();

        // always check HTTP response code first

**if** (responseCode == HttpURLConnection.HTTP\_OK) {

            String disposition = httpConn.getHeaderField("Content-Disposition");

            String contentType = httpConn.getContentType();

            contentLength = httpConn.getContentLength();

**if** (disposition != **null**) {

                // extracts file name from header field

**int** index = disposition.indexOf("filename=");

**if** (index > 0) {

                    fileName = disposition.substring(index + 10,

                            disposition.length() - 1);

                }

            } **else** {

                // extracts file name from URL

                fileName = fileURL.substring(fileURL.lastIndexOf("/") + 1,

                        fileURL.length());

            }

            // output for debugging purpose only

            System.out.println("Content-Type = " + contentType);

            System.out.println("Content-Disposition = " + disposition);

            System.out.println("Content-Length = " + contentLength);

            System.out.println("fileName = " + fileName);

            // opens input stream from the HTTP connection

            inputStream = httpConn.getInputStream();

        } **else** {

**throw** **new** IOException(

                    "No file to download. Server replied HTTP code: "

                            + responseCode);

        }

    }

**public** **void** disconnect() **throws** IOException {

        inputStream.close();

        httpConn.disconnect();

    }

**public** String getFileName() {

**return** **this**.fileName;

    }

**public** **int** getContentLength() {

**return** **this**.contentLength;

    }

**public** InputStream getInputStream() {

**return** **this**.inputStream;

    }

}

This utility class does not actually download the file. It only makes a connection, parses HTTP headers to get file information, and opens the connection’s input stream which will be used by the DownloadTask class.

## 2. Code of the DownloadTask class

**package** net.codejava.swing.download;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** javax.swing.JOptionPane;

**import** javax.swing.SwingWorker;

/\*\*

 \* Execute file download in a background thread and update the progress.

 \* @author www.codejava.net

 \*

 \*/

**public** **class** DownloadTask **extends** SwingWorker<Void, Void> {

**private** **static** **final** **int** BUFFER\_SIZE = 4096;

**private** String downloadURL;

**private** String saveDirectory;

**private** SwingFileDownloadHTTP gui;

**public** DownloadTask(SwingFileDownloadHTTP gui, String downloadURL, String saveDirectory) {

**this**.gui = gui;

**this**.downloadURL = downloadURL;

**this**.saveDirectory = saveDirectory;

    }

    /\*\*

     \* Executed in background thread

     \*/

    @Override

**protected** Void doInBackground() **throws** Exception {

**try** {

            HTTPDownloadUtil util = **new** HTTPDownloadUtil();

            util.downloadFile(downloadURL);

            // set file information on the GUI

            gui.setFileInfo(util.getFileName(), util.getContentLength());

            String saveFilePath = saveDirectory + File.separator + util.getFileName();

            InputStream inputStream = util.getInputStream();

            // opens an output stream to save into file

            FileOutputStream outputStream = **new** FileOutputStream(saveFilePath);

**byte**[] buffer = **new** **byte**[BUFFER\_SIZE];

**int** bytesRead = -1;

**long** totalBytesRead = 0;

**int** percentCompleted = 0;

**long** fileSize = util.getContentLength();

**while** ((bytesRead = inputStream.read(buffer)) != -1) {

                outputStream.write(buffer, 0, bytesRead);

                totalBytesRead += bytesRead;

                percentCompleted = (**int**) (totalBytesRead \* 100 / fileSize);

                setProgress(percentCompleted);

            }

            outputStream.close();

            util.disconnect();

        } **catch** (IOException ex) {

            JOptionPane.showMessageDialog(gui, "Error downloading file: " + ex.getMessage(),

                    "Error", JOptionPane.ERROR\_MESSAGE);

            ex.printStackTrace();

            setProgress(0);

            cancel(**true**);

        }

**return** **null**;

    }

    /\*\*

     \* Executed in Swing's event dispatching thread

     \*/

    @Override

**protected** **void** done() {

**if** (!isCancelled()) {

            JOptionPane.showMessageDialog(gui,

                    "File has been downloaded successfully!", "Message",

                    JOptionPane.INFORMATION\_MESSAGE);

        }

    }

}

## This class implements the javax.swing.SwingWorker class so its doInBackground() method will be executed in a separate thread. It utilizes the HTTPDownloadUtil class to perform the file download, updates file information to the GUI and notifies download progress so the GUI will update the progress bar accordingly.

## 3. Code of the SwingFileDownloadHTTP class

**package** net.codejava.swing.download;

**import** java.awt.Dimension;

**import** java.awt.GridBagConstraints;

**import** java.awt.GridBagLayout;

**import** java.awt.Insets;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.beans.PropertyChangeEvent;

**import** java.beans.PropertyChangeListener;

**import** javax.swing.JButton;

**import** javax.swing.JFileChooser;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.JOptionPane;

**import** javax.swing.JProgressBar;

**import** javax.swing.JTextField;

**import** javax.swing.SwingUtilities;

**import** javax.swing.UIManager;

**import** net.codejava.swing.JFilePicker;

/\*\*

 \* A Swing application that downloads file from an HTTP server.

 \* @author www.codejava.net

 \*

 \*/

**public** **class** SwingFileDownloadHTTP **extends** JFrame **implements**

        PropertyChangeListener {

**private** JLabel labelURL = **new** JLabel("Download URL: ");

**private** JTextField fieldURL = **new** JTextField(30);

**private** JFilePicker filePicker = **new** JFilePicker("Save in directory: ",

            "Browse...");

**private** JButton buttonDownload = **new** JButton("Download");

**private** JLabel labelFileName = **new** JLabel("File name: ");

**private** JTextField fieldFileName = **new** JTextField(20);

**private** JLabel labelFileSize = **new** JLabel("File size (bytes): ");

**private** JTextField fieldFileSize = **new** JTextField(20);

**private** JLabel labelProgress = **new** JLabel("Progress:");

**private** JProgressBar progressBar = **new** JProgressBar(0, 100);

**public** SwingFileDownloadHTTP() {

**super**("Swing File Download from HTTP server");

        // set up layout

        setLayout(**new** GridBagLayout());

        GridBagConstraints constraints = **new** GridBagConstraints();

        constraints.anchor = GridBagConstraints.WEST;

        constraints.insets = **new** Insets(5, 5, 5, 5);

        // set up components

        filePicker.setMode(JFilePicker.MODE\_SAVE);

        filePicker.getFileChooser().setFileSelectionMode(JFileChooser.DIRECTORIES\_ONLY);

        buttonDownload.addActionListener(**new** ActionListener() {

            @Override

**public** **void** actionPerformed(ActionEvent event) {

                buttonDownloadActionPerformed(event);

            }

        });

        fieldFileName.setEditable(**false**);

        fieldFileSize.setEditable(**false**);

        progressBar.setPreferredSize(**new** Dimension(200, 30));

        progressBar.setStringPainted(**true**);

        // add components to the frame

        constraints.gridx = 0;

        constraints.gridy = 0;

        add(labelURL, constraints);

        constraints.gridx = 1;

        constraints.fill = GridBagConstraints.HORIZONTAL;

        constraints.weightx = 1.0;

        add(fieldURL, constraints);

        constraints.gridx = 0;

        constraints.gridy = 1;

        constraints.weightx = 0.0;

        constraints.gridwidth = 2;

        constraints.fill = GridBagConstraints.NONE;

        add(filePicker, constraints);

        constraints.gridy = 2;

        constraints.anchor = GridBagConstraints.CENTER;

        add(buttonDownload, constraints);

        constraints.gridx = 0;

        constraints.gridy = 3;

        constraints.gridwidth = 1;

        constraints.anchor = GridBagConstraints.WEST;

        add(labelFileName, constraints);

        constraints.gridx = 1;

        add(fieldFileName, constraints);

        constraints.gridy = 4;

        constraints.gridx = 0;

        add(labelFileSize, constraints);

        constraints.gridx = 1;

        add(fieldFileSize, constraints);

        constraints.gridx = 0;

        constraints.gridy = 5;

        constraints.gridwidth = 1;

        constraints.anchor = GridBagConstraints.WEST;

        add(labelProgress, constraints);

        constraints.gridx = 1;

        constraints.weightx = 1.0;

        constraints.fill = GridBagConstraints.HORIZONTAL;

        add(progressBar, constraints);

        pack();

        setLocationRelativeTo(**null**);    // center on screen

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

    }

    /\*\*

     \* handle click event of the Upload button

     \*/

**private** **void** buttonDownloadActionPerformed(ActionEvent event) {

        String downloadURL = fieldURL.getText();

        String saveDir = filePicker.getSelectedFilePath();

        // validate input first

**if** (downloadURL.equals("")) {

            JOptionPane.showMessageDialog(**this**, "Please enter download URL!",

                    "Error", JOptionPane.ERROR\_MESSAGE);

            fieldURL.requestFocus();

**return**;

        }

**if** (saveDir.equals("")) {

            JOptionPane.showMessageDialog(**this**,

                    "Please choose a directory save file!", "Error",

                    JOptionPane.ERROR\_MESSAGE);

**return**;

        }

**try** {

            progressBar.setValue(0);

            DownloadTask task = **new** DownloadTask(**this**, downloadURL, saveDir);

            task.addPropertyChangeListener(**this**);

            task.execute();

        } **catch** (Exception ex) {

            JOptionPane.showMessageDialog(**this**,

                    "Error executing upload task: " + ex.getMessage(), "Error",

                    JOptionPane.ERROR\_MESSAGE);

        }

    }

**void** setFileInfo(String name, **int** size) {

        fieldFileName.setText(name);

        fieldFileSize.setText(String.valueOf(size));

    }

    /\*\*

     \* Update the progress bar's state whenever the progress of download changes.

     \*/

    @Override

**public** **void** propertyChange(PropertyChangeEvent evt) {

**if** (evt.getPropertyName().equals("progress")) {

**int** progress = (Integer) evt.getNewValue();

            progressBar.setValue(progress);

        }

    }

    /\*\*

     \* Launch the application

     \*/

**public** **static** **void** main(String[] args) {

**try** {

            // set look and feel to system dependent

            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

        } **catch** (Exception ex) {

            ex.printStackTrace();

        }

        SwingUtilities.invokeLater(**new** Runnable() {

            @Override

**public** **void** run() {

**new** SwingFileDownloadHTTP().setVisible(**true**);

            }

        });

    }

}

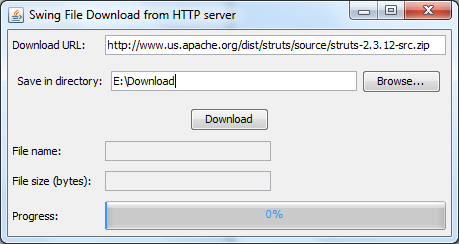
This builds the user interface for the application and wires all the pieces together to form a complete application. To update the progress bar while the download is taking place, it “listens” to the property named “progress” which is repeatedly updated by the DownloadTask class.

## 4. Testing the application

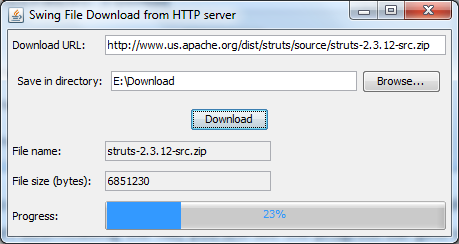
To test this application, we need to have a downloadable URL, for example:

*http://www.us.apache.org/dist/struts/source/struts-2.3.12-src.zip*

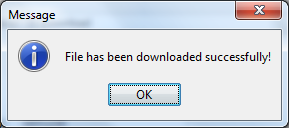
Run the application and type the above URL into the *Download URL* field:



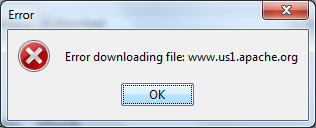
Click **Browse…** button to select a directory where the file will be stored, for example: *E:\Download*, and hit **Download** button to start downloading the file. Just a second you will see the file information is updated in the fields *File name* and *File size*, as well as the progress bar gets updated quickly:



When the progress reaches to 100%, a message dialog appears:



The following error message dialog will appear in case of error such as the URL is invalid:



To experiment yourself with this nice application, download full source code and executable jar file of in the attachments section. You can also clone the [sample project on GitHub](https://github.com/codejava-official/swing-http-file-download).

<https://downloads.apache.org/struts/2.5.33/struts-2.5.33-all.zip>

<https://www.codejava.net/coding/swing-application-to-download-files-from-http-server-with-progress-bar>

(package net.codejava.swing;

import java.awt.FlowLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

import javax.swing.JFileChooser;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextField;

public class JFilePicker extends JPanel {

private String textFieldLabel;

private String buttonLabel;

private JLabel label;

private JTextField textField;

private JButton button;

private JFileChooser fileChooser;

private int mode;

public static final int MODE\_OPEN = 1;

public static final int MODE\_SAVE = 2;

public JFilePicker(String textFieldLabel, String buttonLabel) {

this.textFieldLabel = textFieldLabel;

this.buttonLabel = buttonLabel;

fileChooser = new JFileChooser();

setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));

// creates the GUI

label = new JLabel(textFieldLabel);

textField = new JTextField(30);

button = new JButton(buttonLabel);

button.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent evt) {

buttonActionPerformed(evt);

}

});

add(label);

add(textField);

add(button);

}

private void buttonActionPerformed(ActionEvent evt) {

if (mode == MODE\_OPEN) {

if (fileChooser.showOpenDialog(this) == JFileChooser.APPROVE\_OPTION) {

textField.setText(fileChooser.getSelectedFile().getAbsolutePath());

}

} else if (mode == MODE\_SAVE) {

if (fileChooser.showSaveDialog(this) == JFileChooser.APPROVE\_OPTION) {

textField.setText(fileChooser.getSelectedFile().getAbsolutePath());

}

}

}

public void addFileTypeFilter(String extension, String description) {

FileTypeFilter filter = new FileTypeFilter(extension, description);

fileChooser.addChoosableFileFilter(filter);

}

public void setMode(int mode) {

this.mode = mode;

}

public String getSelectedFilePath() {

return textField.getText();

}

public JFileChooser getFileChooser() {

return this.fileChooser;

}

})