# Very Basic Browser: A Minimalist Web Browser

Borys Bradel

April 13, 2021

## 1 Introduction

The Very Basic Browser is a minimalist web browser. It has only three components: the url field, the HTML text pane, and a status text field. Because of this simplicity, the browser is less than 100 lines of code and fits on a single page.

## 2 Overview

The Very Basic Browser has only three components. The url field is at the top. The HTML text pane is in the center. And the status text field is on the bottom.

The Very Basic Browser has to perform several operations:

- Create the window.
- Populate the window with GUI components.
- Follow links that are selected on the HTML text pane.
- Update the HTML text pane with contents downloaded from the web based on the url field or the followed links.

The browser's code is divided into methods that correspond to these operations.

## 3 Operation Instructions

You can create a new project in an IDE and then add the file to the project. For example, in IntelliJ you can create a new project (File > New > Project...). Then choose a new Java project and give it a name. After that you can drag and drop the file to the src directory. Finally, you need to add the jsoup library. Open Project Structure (File > Project Structure...) and go to the Modules pane. Under dependencies, press the plus (add) button and choose the first option: JARs or Directories. You can find the jsoup jar file in the JetBrains/IntelliJ/lib directory and add it. After that, you can run the browser from the IDE.

For command-line use, you have to put VeryBasicBrowser.java and the jsoup jar file in one directory. Then compile and run the browser in that directory.

The instructions for operating systems other than Windows are the following:

- `/path/to/javac -cp jsoup-1.13.1.jar VeryBasicBrowser.java`
- `/path/to/java -cp .:jsoup-1.13.1.jar VeryBasicBrowser`

The instructions for Windows are the following:

- `C:\path\to\javac -cp jsoup-1.13.1.jar VeryBasicBrowser.java`
- `C:\path\to\java -cp .;jsoup-1.13.1.jar VeryBasicBrowser`

Once the browser is running, you can enter urls and look at websites. Close the browser's window to shut down the browser.

## 4 Implementation

The following is a description of the implementation of the browser. The full source code for the browser is in Listing 1.

The browser is implemented in Java and uses Swing to create the GUI. Swing is not thread safe. All updates need to be done in the Abstract Window Toolkit (AWT) thread. Therefore the main method calls SwingUtilities.invokeLater() with a method reference as a parameter. This method reference is to a method that creates the frame for the browser.

Listing 1: Source Code

```
import org.jsoup.Jsoup;
import org.jsoup.safety.Cleaner;
import org.jsoup.safety.Whitelist;

import javax.swing.*;
import javax.swing.event.HyperlinkEvent;
import javax.swing.event.HyperlinkListener;
import javax.swing.text.html.HTMLEditorKit;
import java.awt.*;
import java.io.IOException;
import java.util.concurrent.
    ExecutionException;

public class VeryBasicBrowser extends JPanel
     implements HyperlinkListener {
  public static void main(String[] args) {
    SwingUtilities.invokeLater(
    VeryBasicBrowser::startGui);
  }

  static void startGui() {
    JFrame frame = new JFrame("Very Basic
    Browser");
    frame.add(new VeryBasicBrowser());
    frame.setDefaultCloseOperation(JFrame.
    EXIT_ON_CLOSE);
    frame.setSize(new Dimension(600, 400));
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
  }

  JTextField urlField, statusField;
  JEditorPane editorPane;

  VeryBasicBrowser() {
    super(new BorderLayout());
    urlField = new JTextField(80);
    urlField.addActionListener(e ->
    urlUpdate(e.getActionCommand()));
    urlField.setFocusAccelerator('L');
    add(urlField, BorderLayout.PAGE_START);
    HTMLEditorKit kit = new HTMLEditorKit();
    kit.setAutoFormSubmission(false);
    editorPane = new JEditorPane("text/html"
    , "");
    editorPane.addHyperlinkListener(this);
    editorPane.setEditable(false);
    editorPane.setEditorKit(kit);
    JScrollPane scrollPane = new JScrollPane
    (editorPane);
    add(scrollPane, BorderLayout.CENTER);
    statusField = new JTextField();
    statusField.setEditable(false);
    add(statusField, BorderLayout.PAGE_END);
  }

  @Override
  public void hyperlinkUpdate(HyperlinkEvent
     event) {
    String url = event.getURL().toString();
    if (event.getEventType() ==
    HyperlinkEvent.EventType.ACTIVATED) {
      urlField.setText(url);
      urlUpdate(url);
    }
    statusField.setText((event.getEventType
    () == HyperlinkEvent.EventType.EXITED) ?
     "" : url);
  }

  void urlUpdate(String url) {
    final String newUrl;
    if (!url.startsWith("http://") && !url.
    startsWith("https://")) {
      newUrl = "http://" + url;
      urlField.setText(newUrl);
    } else {
      newUrl = url;
    }
    SwingWorker<String, Void> updater = new
    SwingWorker<>() {
      @Override
      public String doInBackground() {
        String result = "";
        try {
          org.jsoup.nodes.Document soupDoc =
     Jsoup.connect(newUrl).get();
          result = new Cleaner(Whitelist.
    relaxed()).clean(soupDoc).html();
        } catch (IOException e) {
          statusField.setText(String.format(
    "Malformed url: |%s|.", newUrl));
        }
        return result;
      }
      @Override
      public void done() {
        try {
          String text = get();
          if (text.isEmpty()) {
            statusField.setText(String.
    format("Nothing returned for the url |%s
    |.", newUrl));
          }
          editorPane.setText(text);
          editorPane.setCaretPosition(0);
        } catch (InterruptedException e) {
          statusField.setText(String.format(
    "Something interrupted getting the url
    |%s|.\n", url));
        } catch (ExecutionException e) {
          statusField.setText(String.format(
    "An execution error occurred when
    getting the url |%s|.\n", url));
        }
      }
    };
    updater.execute();
  }
}
```

The browser class extends the JPanel class and is added as a panel to the frame. The browser uses a border layout that allows adding components to the top, center, left, right, and bottom of the panel. The browser has a url text field at the top, the HTML text pane in the center, and a status text field at the bottom. Since the left and right side are not used, the center section expands horizontally.

The browser object acts as a hyperlink listener for the HTML text pane. Therefore, the browser overrides the hyperlinkUpdate() method. The listener reacts to three types of events: ENTERED, EXITED, and ACTIVATED. If the link is activated, the url field is updated with the url and the contents of the link are loaded into the HTML text pane. If the link is exited, the status field is cleared. Otherwise the field is filled in with the url of the link.

The loading requires two steps. First, the web page needs to be downloaded. Second, the downloaded data needs to be put into the text pane. Since the downloading may take a long time and may be blocking, a separate thread needs to be used. The SwingWorker class is used for this purpose. A subclass is created that overrides the doInBackground () and done() methods.

The doInBackground() method is executed asynchronously on a worker thread that is separate from the AWT thread. Therefore the operations in the method can be blocking and time consuming.

The doInBackground() method uses the Jsoup.connect () method to download the specified web page and jsoup's Cleaner class to simplify the downloaded HTML. This simplification helps the underlying HTML text pane class, Java Swing's HTMLEditorKit, to display the HTML properly.

The combination of a library to get the HTML and a text pane to display the HTML can be used in other languages as well. The following are some examples.

- In Python, urllib and Beautiful Soup can be used to get the HTML and Qt's QTextBrowser can be used to show the HTML in the GUI.
- In C/C++, libcurl can be used to get the HTML and Qt's QTextBrowser can be used to show the HTML in the GUI.
- In Kotlin on Android, you can use jsoup to get the HTML and TextView to show the HTML in the GUI.

The resulting data from Jsoup.connect() is saved to be passed to the done() method. The done() method is executed in the AWT thread after doInBackground () is finished. The done() method fills in the HTML text pane with the downloaded text.

# 5 Conclusion

This paper introduces a minimalist web browser. Because of the browser's simplicity, the code is short enough to fit on a single page. With some additions, the browser could be made an even more useful replacement for viewing simple web pages.