Joel Edwards
Course: Java Programming 1
Homework 9
May 22, 2011

9.A)

The shortest and longest word logic reports the first word found that has the shortest or longest length.

Source:

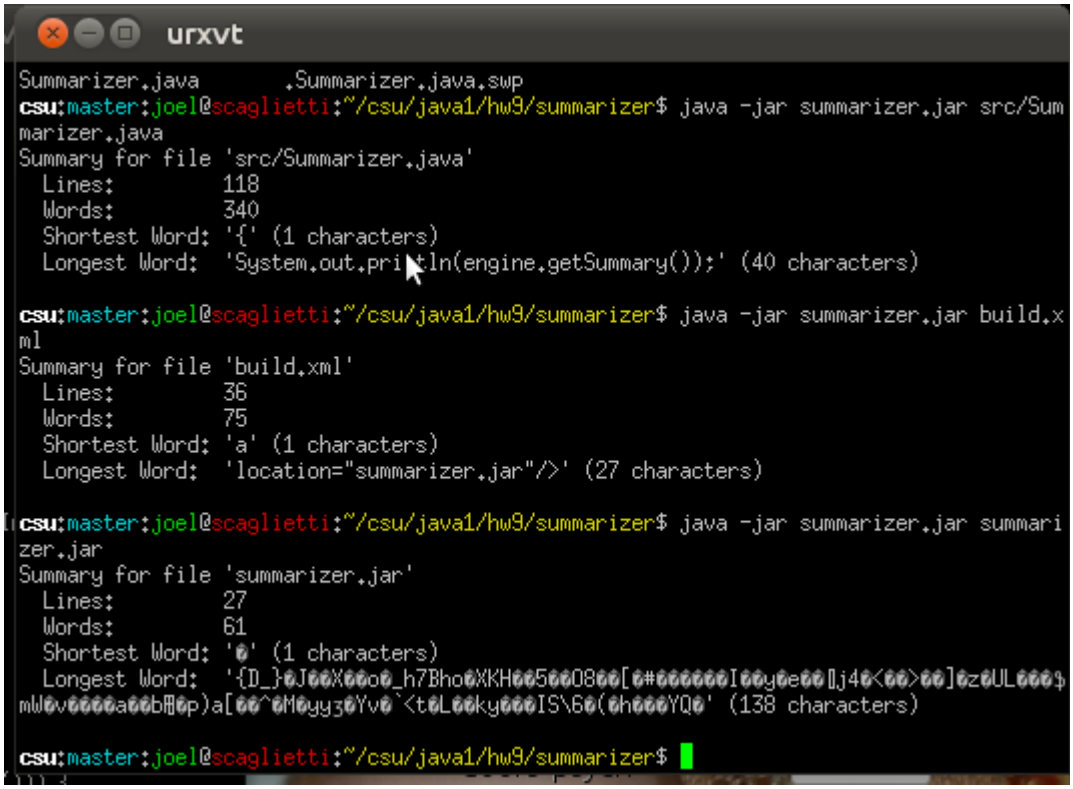
CalculatorGUI.java:

```
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.io.IOException;
import java.util.StringTokenizer;

public class Summarizer
{
 // Program Static Methods
    public static void main(String[] args)
    {
        if (args.length < 1) {
            usage("No file name supplied.");
        }
        if (args.length > 1) {
            usage("Too many arguments.");
        }

        File file = new File(args[0]);
        if (!file.exists()) {
            error("File '" +file+ "' does not exist.");
        }
        if (!file.isFile()) {
            error("Path '" +file+ "' does not refer to a normal
file.");
        }
        if (!file.canRead()) {
            error("File '" +file+ "' is not readable.");
        }

        try {
            Summarizer engine = new Summarizer(file);
            engine.generateSummary();
            System.out.println(engine.getSummary());
        } catch (IOException e) {
            error("Reading of file '" +file+  "' failed.");
```

```
        }
    }

    public static void usage()
    {
        error(null, true);
    }

    public static void usage(String message)
    {
        error(message, true);
    }

    public static void error(String message)
    {
        error(message, false);
    }

    public static void error(String message, boolean showUsage)
    {
        if ((message != null) && (message.length() > 0)) {
            System.out.println("E: " + message);
        }
        if (showUsage) {
            System.out.println("Usage: summarizer <file_name>");
            System.out.println("       file_name - the text file to
summarize");
        }
        System.exit(1);
    }

 // Class Data
    private StringBuffer summary;
    private File file;

    private String longestWord = null;
    private String shortestWord = null;
    private int lineCount = 0;
    private int wordCount = 0;

 // Class Methods
    public Summarizer(File file)
    {
        summary = new StringBuffer();
        this.file = file;
    }

    public void generateSummary()
    throws IOException
```

```
    {
        BufferedReader reader = new BufferedReader(new
InputStreamReader(new DataInputStream(new FileInputStream(file))));
        StringTokenizer tokenizer = null;
        String line = reader.readLine();
        String token = null;
        while (line != null) {
            tokenizer = new StringTokenizer(line);
            lineCount++;
            while (tokenizer.hasMoreTokens()) {
                token = tokenizer.nextToken();
                if ((longestWord == null) || (longestWord.length() <
token.length())) {
                    longestWord = token;
                }
                if ((shortestWord == null) || (shortestWord.length()
> token.length())) {
                    shortestWord = token;
                }
                wordCount++;
            }
            line = reader.readLine();
        }

        summary.append("Summary for file '" +file+ "'\n");
        summary.append("  Lines:         " +lineCount+ "\n");
        summary.append("  Words:         " +wordCount+ "\n");
        summary.append("  Shortest Word: '" +shortestWord+ "' ("
+shortestWord.length()+ " characters)\n");
        summary.append("  Longest Word:  '" +longestWord+ "' ("
+longestWord.length()+ " characters)\n");
    }

    public String getSummary()
    {
        return summary.toString();
    }
}
```

Test Output:


urxvt
Summarizer.java .Summarizer.java.swp
csu:master:joel@scaglietti:~/csu/java1/hw9/summarizer$ java -jar summarizer.jar src/Summarizer.java
Summary for file 'src/Summarizer.java'
Lines: 118
Words: 340
Shortest Word: '{' (1 characters)
Longest Word: 'System.out.println(engine.getSummary());' (40 characters)
csu:master:joel@scaglietti:~/csu/java1/hw9/summarizer$ java -jar summarizer.jar build.xml
Summary for file 'build.xml'
Lines: 36
Words: 75
Shortest Word: 'a' (1 characters)
Longest Word: 'location="summarizer.jar"/>' (27 characters)
csu:master:joel@scaglietti:~/csu/java1/hw9/summarizer$ java -jar summarizer.jar summarizer.jar
Summary for file 'summarizer.jar'
Lines: 27
Words: 61
Shortest Word: '' (1 characters)
Longest Word: '
(138 characters)
csu:master:joel@scaglietti:~/csu/java1/hw9/summarizer$

9.B)

I expanded on this assignment a lot in order to get a better understanding of the javax.swing package. I kept the *load* and *save* buttons, although they are not necessary with the provided implementation.

I have added *File* and *Edit* menus with mnemonics (accelerators), as well as keyboard shortcuts. The interface updates to only allow those operations which are necessary for the given state.

The interface is tabbed, allowing the user to add any number of files based upon their needs, and switch between files easily. They can open new tabs without a file name, edit them, then save if desired. Tabs present the user with a dialog if they attempt to close a tab with modified contents. The tab icon (left side of tab; not the close button) indicates the state of the tab. The tab state can be one of Fresh, Modified, or Saved.

It is worth noting that the source alone is not sufficient to build the editor. An archived copy of the project should be provided along with this document. The archive will contain all of the source, and the necessary associated files.

Editor.java:

The Editor class extends the javax.swing.JFrame class, and serves as the main window and entry point for the program. It and the Tab class are heavily interdependent, but the split into these two main classes makes the design cleaner.

```
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.File;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
import javax.swing.JTextArea;
import javax.swing.UIManager;

public class Editor
extends JFrame
```

```
implements ActionListener, ChangeListener, KeyListener
{
    public static void main(String[] args)
    {
        Editor window = new Editor("Editor");
        window.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        window.setVisible(true);
    }

 // Class Components
    private String clipboard = null;
    private String title;
    private File defaultOpenDir = new File(".");
    private File defaultSaveDir = new File(".");
    private Terminator terminator = null;

    private int currentTab = 0;
    private boolean isDoubleBuffered = false;

    private JTabbedPane tabs = new JTabbedPane();

    private JPanel buttonPanel = new JPanel(new BorderLayout());
    private JButton loadButton = new JButton("Load");
    private JButton saveButton = new JButton("Save");

    private JMenuBar menuBar = new JMenuBar();
    private JMenu fileMenu   = new JMenu("File");
    private JMenu editMenu   = new JMenu("Edit");

 // File Menu Items
    private JMenuItem newFileMenuItem    = new JMenuItem("New",
KeyEvent.VK_N);
    private JMenuItem openFileMenuItem   = new JMenuItem("Open",
KeyEvent.VK_O);
    private JMenuItem saveFileMenuItem   = new JMenuItem("Save",
KeyEvent.VK_S);
    private JMenuItem closeFileMenuItem  = new JMenuItem("Close",
KeyEvent.VK_C);
    private JMenuItem saveAsFileMenuItem = new JMenuItem("Save As",
KeyEvent.VK_A);
    private JMenuItem quitFileMenuItem   = new JMenuItem("Quit",
KeyEvent.VK_Q);

 // Edit Menu Items
    private JMenuItem selectAllEditMenuItem = new JMenuItem("Select
All", KeyEvent.VK_A);
    private JMenuItem copyEditMenuItem   = new JMenuItem("Copy",
KeyEvent.VK_C);
    private JMenuItem cutEditMenuItem    = new JMenuItem("Cut",
```

```
KeyEvent.VK_X);
    private JMenuItem pasteEditMenuItem  = new JMenuItem("Paste",
KeyEvent.VK_P);


    public Editor(String title)
    {
        super(title);
        this.title = title;
        setMinimumSize(new Dimension(400, 400));

        tabs.setTabLayoutPolicy(JTabbedPane.SCROLL_TAB_LAYOUT);

        createMenu();
        setJMenuBar(menuBar);

        setLayout(new BorderLayout());
        buttonPanel.add(loadButton, BorderLayout.WEST);
        buttonPanel.add(saveButton, BorderLayout.EAST);
        add(buttonPanel, BorderLayout.SOUTH);
        add(tabs, BorderLayout.CENTER);
        createNewTab(null);
        pack();

        terminator = new Terminator(this, tabs);
        this.addWindowListener(terminator);
        loadButton.addActionListener(this);
        saveButton.addActionListener(this);

     // Make sure keyboard shortcut combinations work everywhere
        this.addKeyListener(this);
        this.getContentPane().addKeyListener(this);
        tabs.addKeyListener(this);
        loadButton.addKeyListener(this);
        saveButton.addKeyListener(this);

        updateGui();
    }

    private void createMenu()
    {
     // File Menu
        fileMenu.setMnemonic(KeyEvent.VK_F);
        menuBar.add(fileMenu);
        fileMenu.add(newFileMenuItem);
        fileMenu.add(openFileMenuItem);
        fileMenu.add(saveFileMenuItem);
        fileMenu.add(saveAsFileMenuItem);
        fileMenu.add(closeFileMenuItem);
```

```
        fileMenu.add(quitFileMenuItem);
        newFileMenuItem.addActionListener(this);
        openFileMenuItem.addActionListener(this);
        saveFileMenuItem.addActionListener(this);
        saveAsFileMenuItem.addActionListener(this);
        closeFileMenuItem.addActionListener(this);
        quitFileMenuItem.addActionListener(this);

     // Edit Menu
        editMenu.setMnemonic(KeyEvent.VK_E);
        menuBar.add(editMenu);
        editMenu.add(selectAllEditMenuItem);
        editMenu.add(copyEditMenuItem);
        editMenu.add(cutEditMenuItem);
        editMenu.add(pasteEditMenuItem);
        selectAllEditMenuItem.addActionListener(this);
        copyEditMenuItem.addActionListener(this);
        cutEditMenuItem.addActionListener(this);
        pasteEditMenuItem.addActionListener(this);
    }

// Tab Operations
    private void createNewTab(File file)
    {
        Tab tab = new Tab(this, tabs, file, isDoubleBuffered);
        tabs.setSelectedIndex(tabs.indexOfComponent(tab));
    }

    private void closeTabAt(int index)
    {
        if (tabs.getTabCount() >= 1) {
            tabs.removeTabAt(index);
        }
    }

    private void closeCurrentTab()
    {
        int index = tabs.getSelectedIndex();
        if (index >= 0) {
            Tab tab = (Tab)tabs.getComponentAt(index);
            tab.close();
        }
    }

    private int getOpenIndex(File file) {
        int total = tabs.getTabCount();
        Tab tab = null;
        for (int i = 0; i < total; i++) {
            tab = (Tab)tabs.getComponentAt(i);
```

```
            if ((tab.getFile() != null) &&
tab.getFile().equals(file)) {
                return i;
            }
        }
        return -1;
    }


 // File Operations
    private void open()
    {
        // select file to load
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setDialogTitle("Open File");
        fileChooser.setCurrentDirectory(defaultOpenDir);
        fileChooser.setMultiSelectionEnabled(false);
        int returnVal = fileChooser.showOpenDialog(this);
        if (returnVal == JFileChooser.APPROVE_OPTION)
        {
            defaultOpenDir = fileChooser.getCurrentDirectory();
            File file = fileChooser.getSelectedFile();

            int openIndex = getOpenIndex(file);
            if (openIndex < 0) {
                Tab tab = (Tab)(tabs.getSelectedComponent());
                if ((tab != null) && tab.isFresh()) {
                    tab.setFile(file);
                } else {
                    createNewTab(file);
                }
            } else {
                tabs.setSelectedIndex(openIndex);
            }
        }
    }

    private void save()
    {
        int index = tabs.getSelectedIndex();
        if (index < 0) {
            return;
        }

        Tab tab = (Tab)tabs.getComponentAt(index);
        if (tab.getFile() == null) {
            saveAs();
        } else {
            tab.save();
```

```
        }
    }

    private void saveAs()
    {
        int index = tabs.getSelectedIndex();
        if (index < 0) {
            return;
        }

        Tab tab = (Tab)tabs.getComponentAt(index);
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setDialogTitle("Save As File");
        fileChooser.setCurrentDirectory(defaultSaveDir);
        fileChooser.setMultiSelectionEnabled(false);
        int returnVal = fileChooser.showSaveDialog(this);
        if (returnVal == JFileChooser.APPROVE_OPTION)
        {
            defaultSaveDir = fileChooser.getCurrentDirectory();
            File file = fileChooser.getSelectedFile();
            tab.saveToFile(file);
        }
    }

// Edit Operations
    private void selectAll()
    {
        int index = tabs.getSelectedIndex();
        if (index < 0) {
            return;
        }
        Tab tab = (Tab)tabs.getComponentAt(index);
        tab.getArea().selectAll();
    }

    private static int EDIT_OPERATION_COPY  = 1;
    private static int EDIT_OPERATION_CUT   = 2;
    private static int EDIT_OPERATION_PASTE = 3;

    private void editOperation(int operation) {
        int index = tabs.getSelectedIndex();
        if (index >= 0) {
            Tab tab = (Tab)tabs.getComponentAt(index);
            JTextArea area = tab.getArea();
            int selectionStart = area.getSelectionStart();
            int selectionEnd = area.getSelectionEnd();
            int selectionWidth = selectionEnd - selectionStart;
            int caretPosition = area.getCaretPosition();
            if (selectionWidth > 0) {
```

```
                if ((operation == EDIT_OPERATION_COPY) ||
                    (operation == EDIT_OPERATION_CUT)) {
                    clipboard = area.getSelectedText();
                    if (operation == EDIT_OPERATION_CUT) {
                        area.replaceSelection("");
                    }
                } else if (operation == EDIT_OPERATION_PASTE) {
                    area.replaceSelection(clipboard);
                }
            } else if ((operation == EDIT_OPERATION_PASTE) &&
                       (caretPosition >= 0)) {
                area.insert(clipboard, caretPosition);
            }
        }
    }

    private void copy()
    {
        editOperation(EDIT_OPERATION_COPY);
    }

    private void cut()
    {
        editOperation(EDIT_OPERATION_CUT);
    }

    private void paste()
    {
        editOperation(EDIT_OPERATION_PASTE);
    }

// Quit the Application
    private void quit() {
        terminator.windowClosing(null);
    }

// Keep the GUI up-to-date after various events
    public void updateGui()
    {
        int index = tabs.getSelectedIndex();
        if (index >= 0) {
            Tab tab = (Tab)tabs.getComponentAt(index);
            if (tab.isFresh()) {
                saveFileMenuItem.setEnabled(false);
                saveAsFileMenuItem.setEnabled(false);
            } else {
                saveAsFileMenuItem.setEnabled(true);
                if (tab.isSaved()) {
                    saveFileMenuItem.setEnabled(false);
```

```
                } else {
                    saveFileMenuItem.setEnabled(true);
                }
            }
            closeFileMenuItem.setEnabled(true);

            JTextArea area = tab.getArea();
            int selectionStart = area.getSelectionStart();
            int selectionEnd = area.getSelectionEnd();
            int selectionWidth = selectionEnd - selectionStart;
            if (selectionWidth > 0) {
                cutEditMenuItem.setEnabled(true);
                copyEditMenuItem.setEnabled(true);
            } else {
                cutEditMenuItem.setEnabled(false);
                copyEditMenuItem.setEnabled(false);
            }

            int caretPosition = area.getCaretPosition();
            if (((caretPosition >= 0) || (selectionWidth > 0)) &&
(clipboard != null)) {
                pasteEditMenuItem.setEnabled(true);
            } else {
                pasteEditMenuItem.setEnabled(false);
            }
        } else {
            closeFileMenuItem.setEnabled(false);
            saveFileMenuItem.setEnabled(false);
            saveAsFileMenuItem.setEnabled(false);
        }
    }

 // ActionListener
    public void actionPerformed(ActionEvent evt)
    {
        Object source = evt.getSource();
        if (source == loadButton) {
            open();
        }
        else if (source == saveButton) {
            save();
        }
        else if (source == openFileMenuItem) {
            open();
        }
        else if (source == saveFileMenuItem) {
            save();
        }
        else if (source == saveAsFileMenuItem) {
```

```
            saveAs();
        }
        else if (source == newFileMenuItem) {
            createNewTab(null);
        }
        else if (source == closeFileMenuItem) {
            closeCurrentTab();
        }
        else if (source == quitFileMenuItem) {
            quit();
        }
        else if (source == selectAllEditMenuItem) {
            selectAll();
        }
        else if (source == copyEditMenuItem) {
            copy();
        }
        else if (source == cutEditMenuItem) {
            cut();
        }
        else if (source == pasteEditMenuItem) {
            paste();
        }
        updateGui();
    }

 // StateListener
    public void stateChanged(ChangeEvent evt) {
        Object source = evt.getSource();
        if (source == tabs) {
            updateGui();
        }
    }

 // KeyListener
    public void keyPressed(KeyEvent evt) {
        int modifiers = evt.getModifiersEx();
        int keyCode = evt.getKeyCode();

     // Ctrl + <key>
        int onmask = KeyEvent.CTRL_DOWN_MASK;
        int offmask = KeyEvent.SHIFT_DOWN_MASK |
KeyEvent.ALT_DOWN_MASK;
        if ((modifiers & (onmask | offmask)) == onmask) {
            if (keyCode == KeyEvent.VK_S) {
                save();
            } else if (keyCode == KeyEvent.VK_A) {
                selectAll();
            } else if (keyCode == KeyEvent.VK_Q) {
```

```
                quit();
            } else if (keyCode == KeyEvent.VK_N) {
                createNewTab(null);
            } else if (keyCode == KeyEvent.VK_O) {
                open();
            } else if (keyCode == KeyEvent.VK_W) {
                closeCurrentTab();
            }
        }

     // Ctrl + Shift + <key>
        onmask = KeyEvent.CTRL_DOWN_MASK | KeyEvent.SHIFT_DOWN_MASK;
        offmask = KeyEvent.ALT_DOWN_MASK;
        if ((modifiers & (onmask | offmask)) == onmask) {
            if (keyCode == KeyEvent.VK_S) {
                saveAs();
            }
        }

        updateGui();
    }

    public void keyReleased(KeyEvent evt) {
        updateGui();
    }

    public void keyTyped(KeyEvent evt) {
        updateGui();
    }
}
```

Resources.java:

The Resources class provides access to content stored within the jar generated by Ant.

```
import java.awt.Image;
import javax.swing.ImageIcon;

public class Resources {
    public static ImageIcon getAsImageIcon(String name, int height,
int width) {
        return new ImageIcon((new
ImageIcon(ClassLoader.getSystemResource(name))).getImage().getScaledI
nstance(height, width, Image.SCALE_SMOOTH));
    }
}
```

Tab.java:

The class wraps the concept of a tab around a JTextArea and associated content. It also handles all of the file io.

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.event.CaretEvent;
import javax.swing.event.CaretListener;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextArea;
import javax.swing.UIManager;

public class Tab
extends JPanel
implements ActionListener, CaretListener, ChangeListener,
DocumentListener
{
    private static ImageIcon closeTabIcon  =
Resources.getAsImageIcon("resources/icons/close.png", 10, 10);
    private static ImageIcon modifiedTabIcon =
Resources.getAsImageIcon("resources/icons/modified.png", 16, 16);
    private static ImageIcon freshTabIcon  =
Resources.getAsImageIcon("resources/icons/fresh.png", 16, 16);
    private static ImageIcon savedTabIcon  =
Resources.getAsImageIcon("resources/icons/saved.png", 16, 16);
    private static Dimension closeTabButtonSize = new
Dimension(closeTabIcon.getIconWidth() + 4,
closeTabIcon.getIconHeight() + 4);
    private static int untitledCount = 0;
```

```
    private boolean fresh = true;
    private boolean saved = true;
    private File file = null;
    private Editor parent = null;
    private JTabbedPane pane = null;
    private JPanel tabComponent = null;
    private JLabel tabIcon = null;// for icon FINISH!
    private JLabel tabLabel = null;
    private JButton tabButton = null;
    private JTextArea area = new JTextArea();
    private JScrollPane scrollArea = new JScrollPane(area);


    public Tab(Editor parent, JTabbedPane pane, File file, boolean
isDoubleBuffered)
    {
        super(new BorderLayout(), isDoubleBuffered);
        assert(parent != null);
        this.parent = parent;
        this.pane = pane;
        add(scrollArea, BorderLayout.CENTER);

        String title = "Untitled" + ++untitledCount;
        int index = pane.getTabCount();
        pane.insertTab(null, null, this, "", index);

        tabComponent = new JPanel(new BorderLayout());
        tabIcon = new JLabel(freshTabIcon);
        tabLabel = new JLabel(title);
        tabButton = new JButton(closeTabIcon);
        tabButton.setPreferredSize(closeTabButtonSize);

        tabComponent.add(tabIcon, BorderLayout.WEST);
        tabComponent.add(tabLabel, BorderLayout.CENTER);
        tabComponent.add(tabButton, BorderLayout.EAST);
        pane.setTabComponentAt(index, tabComponent);

        tabButton.addActionListener(this);
        area.getDocument().addDocumentListener(this);
        area.addCaretListener(this);
        area.addKeyListener(parent);

        if (file != null) {
            setFile(file);
        }
    }

    private void setSaved(boolean saved) {
        this.fresh = false;
```

```
        this.saved = saved;
        if (saved) {
            tabIcon.setIcon(savedTabIcon);
        }
        else {
            tabIcon.setIcon(modifiedTabIcon);
        }
        parent.updateGui();
    }

    public boolean isFresh()
    {
        return fresh;
    }

    public boolean isSaved()
    {
        return saved;
    }

    public File getFile()
    {
        return file;
    }

    public JTextArea getArea()
    {
        return area;
    }

    public void setFile(File file)
    {
        File last = this.file;
        this.file = file;
        if (load()) {
            tabLabel.setText(file.getName());
            setSaved(true);
        } else {
            this.file = last;
            load();
        }
    }

    public void saveToFile(File file)
    {
        System.out.println("saveToFile(): " + file);
        boolean write = true;
        if (file.exists()) {
            JOptionPane confirm = new JOptionPane("The file already
```

```
exists. Do you wish to overwrite it?",
                                                    JOptionPane.QUESTIO
N_MESSAGE,
                                                    JOptionPane.YES_NO_
OPTION,
                                                    UIManager.getIcon("
OptionPane.questionIcon"));
            JDialog dialog = confirm.createDialog("Overwrite File");
            dialog.setVisible(true);
            Object selectedValue = confirm.getValue();
            if (selectedValue != null) {
                int value = (Integer)selectedValue;
                if (value != JOptionPane.YES_OPTION) {
                    write = false;
                }
            }
        }
        if (write) {
            File last = this.file;
            this.file = file;
            if (store()) {
                tabLabel.setText(file.getName());
                setSaved(true);
            } else {
                this.file = last;
            }
        }
    }

    public void save() {
        if (store()) {
            setSaved(true);
        }
    }

    private boolean load() {
        boolean result = true;
        try {
            StringBuffer contents = new StringBuffer();
            BufferedReader reader = new BufferedReader(new
FileReader(file));
            char[] buffer = new char[4096];
            int numRead = 0;

            while ((numRead = reader.read(buffer)) != -1) {
                String data = String.valueOf(buffer, 0, numRead);
                contents.append(data);
                buffer = new char[4096];
            }
```

```
            reader.close();

            area.setText(contents.toString());
        } catch (Exception e) {
            result = false;
        }
        return result;
    }

    private boolean store() {
        boolean result = true;
        try {
            BufferedWriter writer = new BufferedWriter(new
FileWriter(file));
            writer.write(area.getText());
            writer.close();
        } catch (Exception e) {
            result = false;
        }
        return true;
    }

    public void close() {
        boolean remove = true;
        if (!isSaved()) {
            JOptionPane confirm = new JOptionPane("This tab's
contents have been modified. Do you wish to close without saving?",
                                                  JOptionPane.QUESTIO
N_MESSAGE,
                                                  JOptionPane.YES_NO_
OPTION,
                                                  UIManager.getIcon("
OptionPane.questionIcon"));
            JDialog dialog = confirm.createDialog("Tab Contents
Modified");
            dialog.setVisible(true);
            Object selectedValue = confirm.getValue();
            if (selectedValue != null) {
                int value = (Integer)selectedValue;
                if (value != JOptionPane.YES_OPTION) {
                    remove = false;
                }
            }
        }
        if (remove) {
            pane.removeTabAt(pane.indexOfComponent(this));
        }
    }
```

```
 // Delegate methods
    public void stateChanged(ChangeEvent evt) {
        Object source = evt.getSource();
        parent.updateGui();
    }

    public void actionPerformed(ActionEvent evt) {
        Object source = evt.getSource();
        if (source == tabButton) {
            close();
        }
        parent.updateGui();
    }

    public void insertUpdate(DocumentEvent evt) {
        setSaved(false);
    }

    public void removeUpdate(DocumentEvent evt) {
        setSaved(false);
    }

    public void changedUpdate(DocumentEvent evt) {
        setSaved(false);
    }

    public void caretUpdate(CaretEvent evt) {
        parent.updateGui();
    }
}
```

Terminator.java:

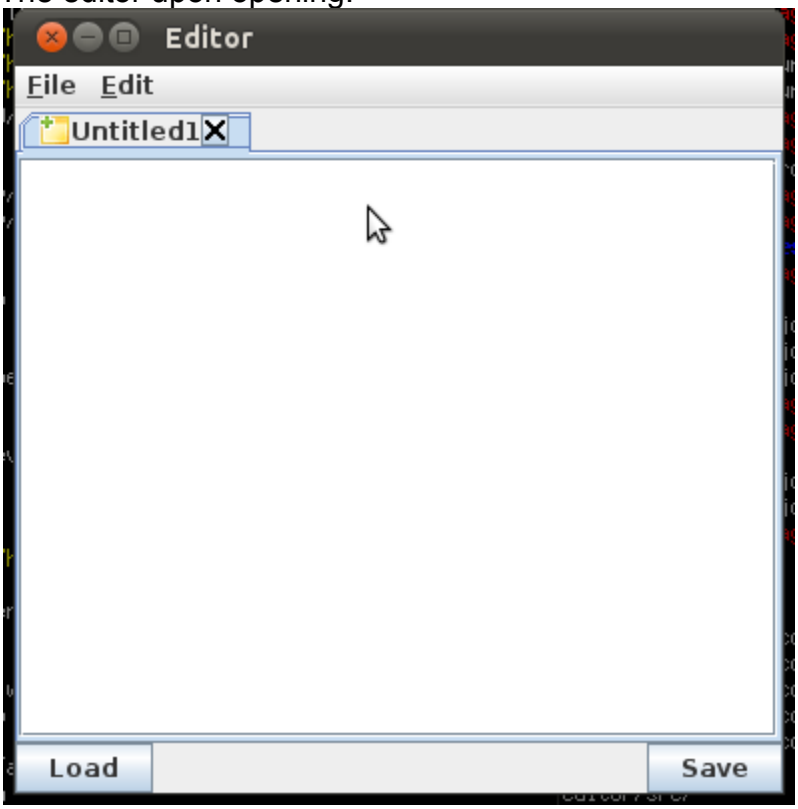
The Terminator class handles the closing logic for the editor, setting up a confirmation dialog if there are any unsaved tabs open in the editor.

```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JDialog;
import javax.swing.JOptionPane;
import javax.swing.JTabbedPane;
import javax.swing.UIManager;

public class Terminator
extends WindowAdapter
{
    private Editor window = null;
    private JTabbedPane tabs = null;
```

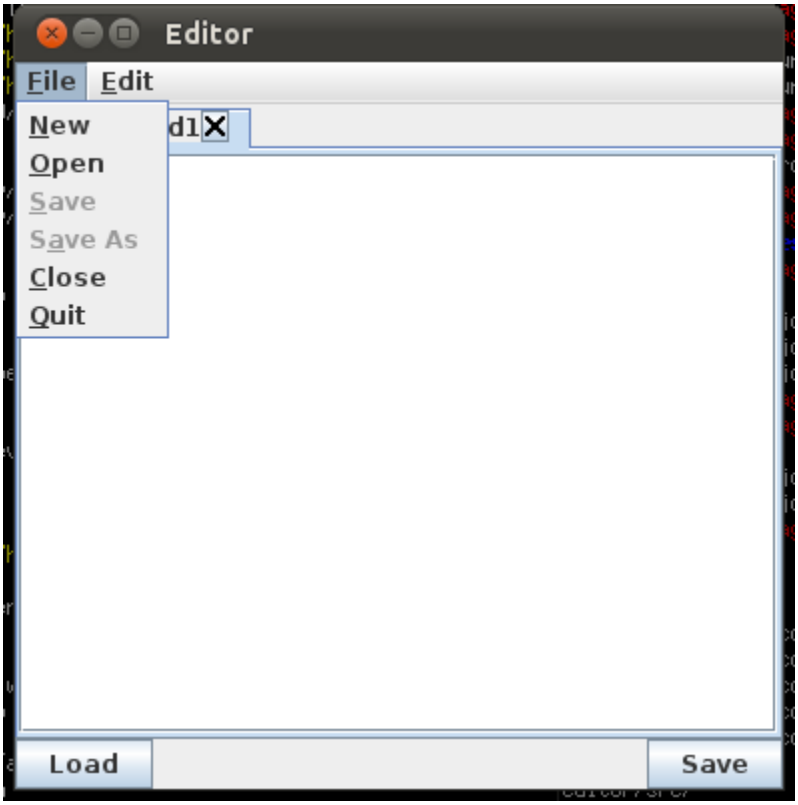
```
    public Terminator(Editor window, JTabbedPane tabs)
    {
        this.window = window;
        this.tabs = tabs;
    }

    private boolean unsavedTabs()
    {
        int total = tabs.getTabCount();
        Tab tab = null;
        for (int i = 0; i < total; i++) {
            tab = (Tab)tabs.getComponentAt(i);
            if (!tab.isSaved() && !tab.isFresh()) {
                return true;
            }
        }
        return false;
    }

    public void windowClosing(WindowEvent evt) {
        boolean close = true;
        if (unsavedTabs()) {
            JOptionPane confirm = new JOptionPane("There are tabs
with modified content. Do you still wish to quit?",
                                                  JOptionPane.QUESTIO
N_MESSAGE,
                                                  JOptionPane.YES_NO_
OPTION,
                                                  UIManager.getIcon("
OptionPane.questionIcon"));
            JDialog dialog = confirm.createDialog("Unsaved Tabs");
            dialog.setVisible(true);
            Object selectedValue = confirm.getValue();
            if (selectedValue != null) {
                int value = (Integer)selectedValue;
                if (value != JOptionPane.YES_OPTION) {
                    close = false;
                }
            }
        }
        if (close) {
            window.setVisible(false);
            System.exit(0);
        }
    }
}
```

Test Output:

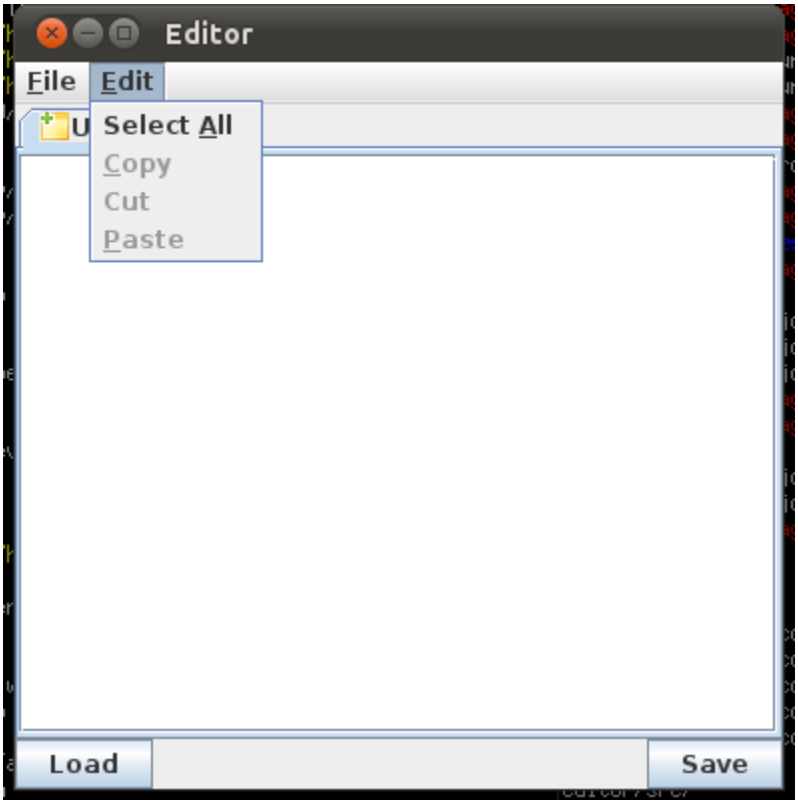
The editor upon opening.



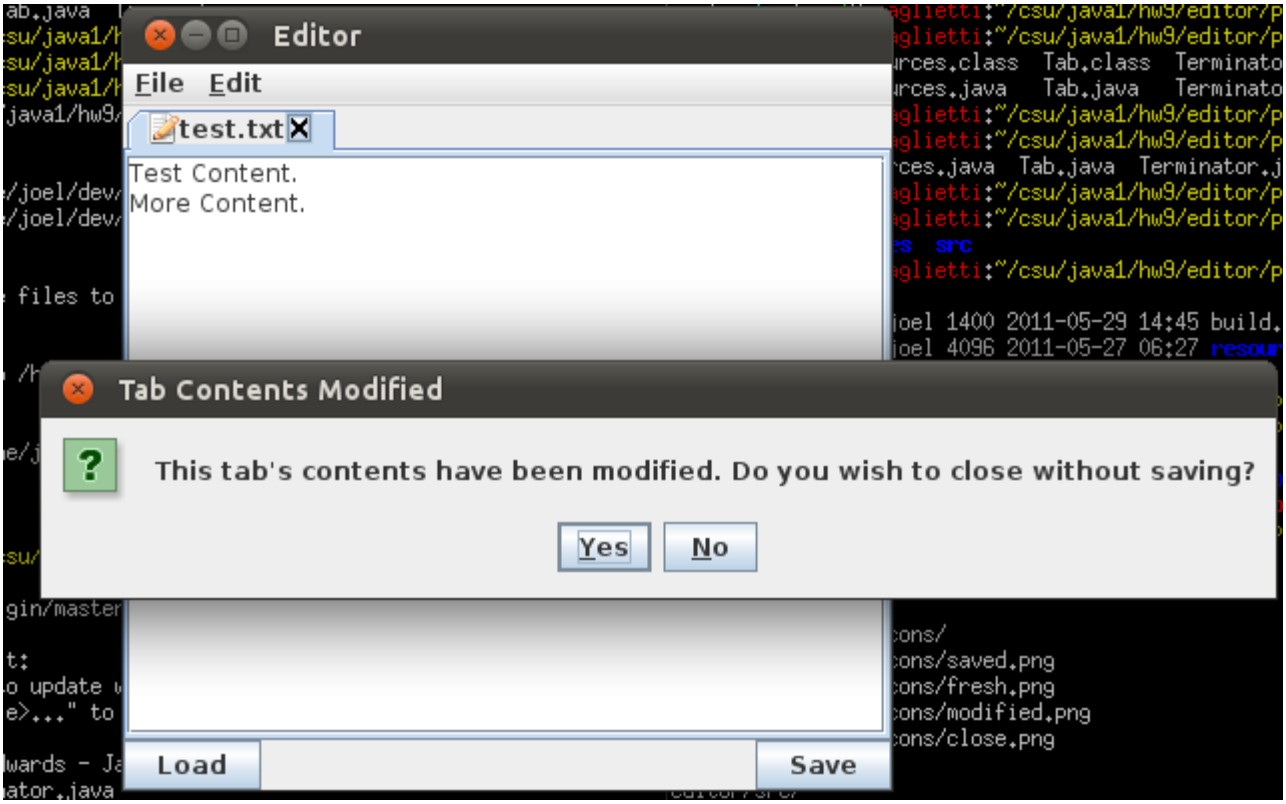
File menu.



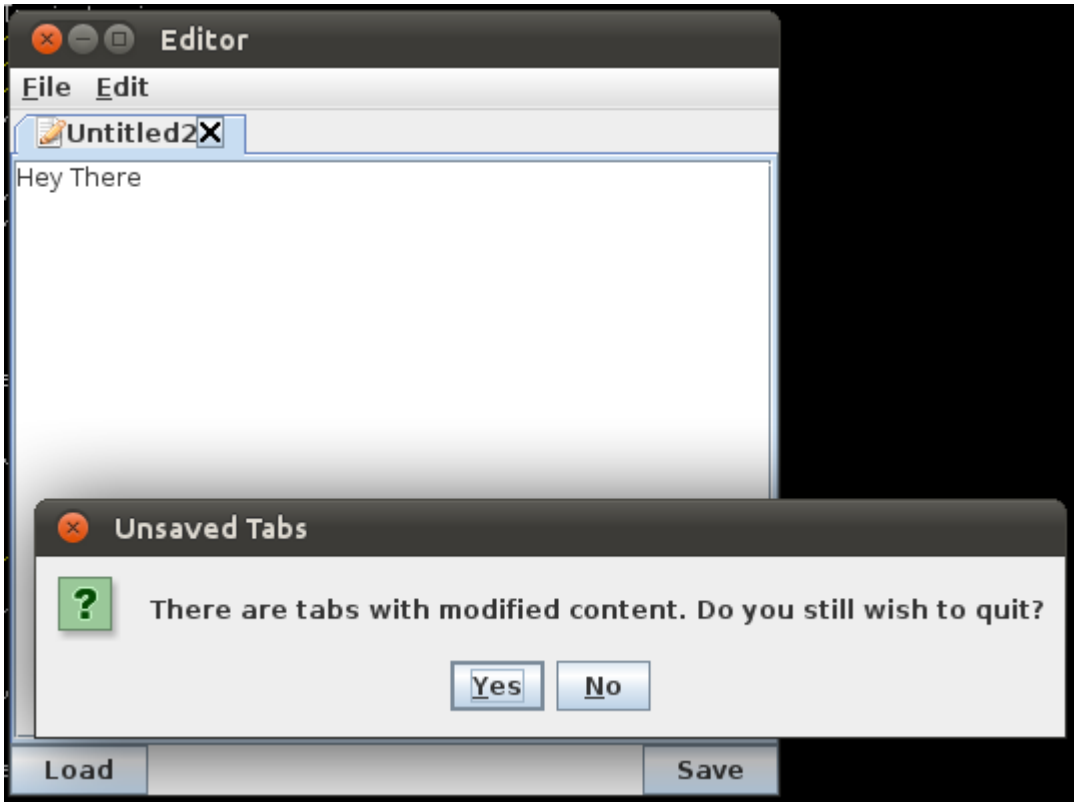
Edit menu.



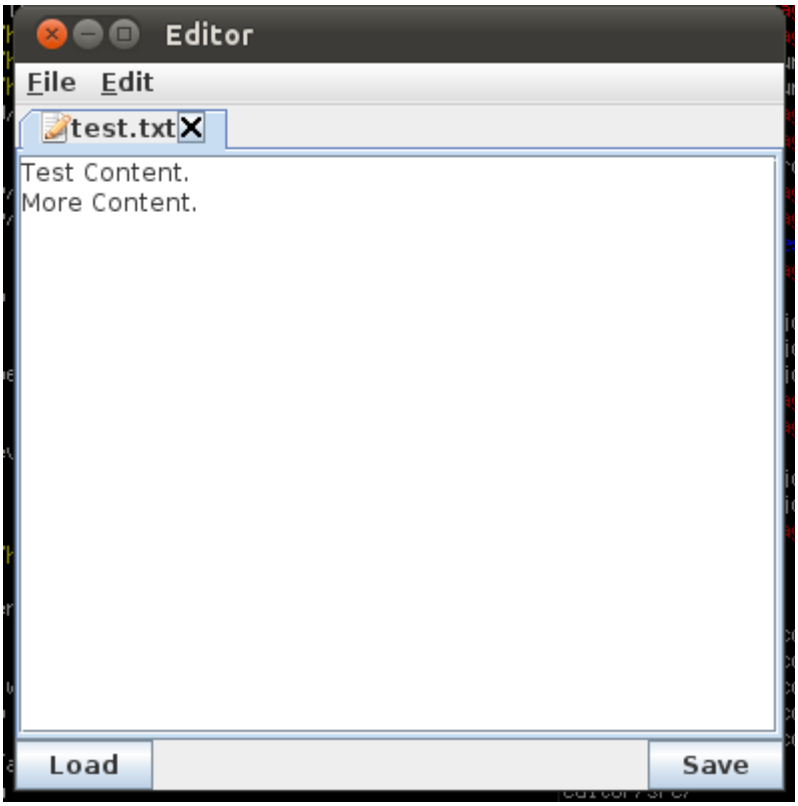
Confirm close tab.



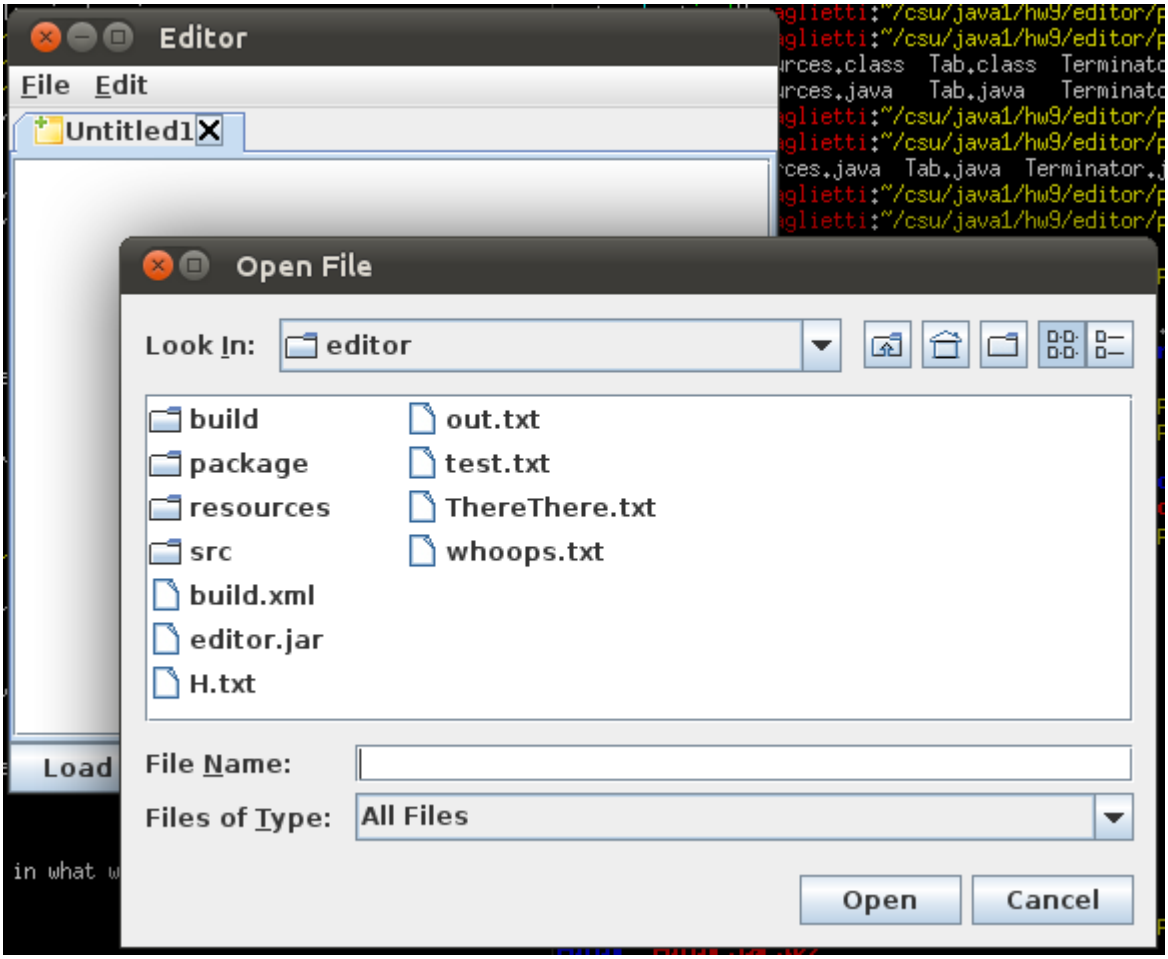
Confirm quit.



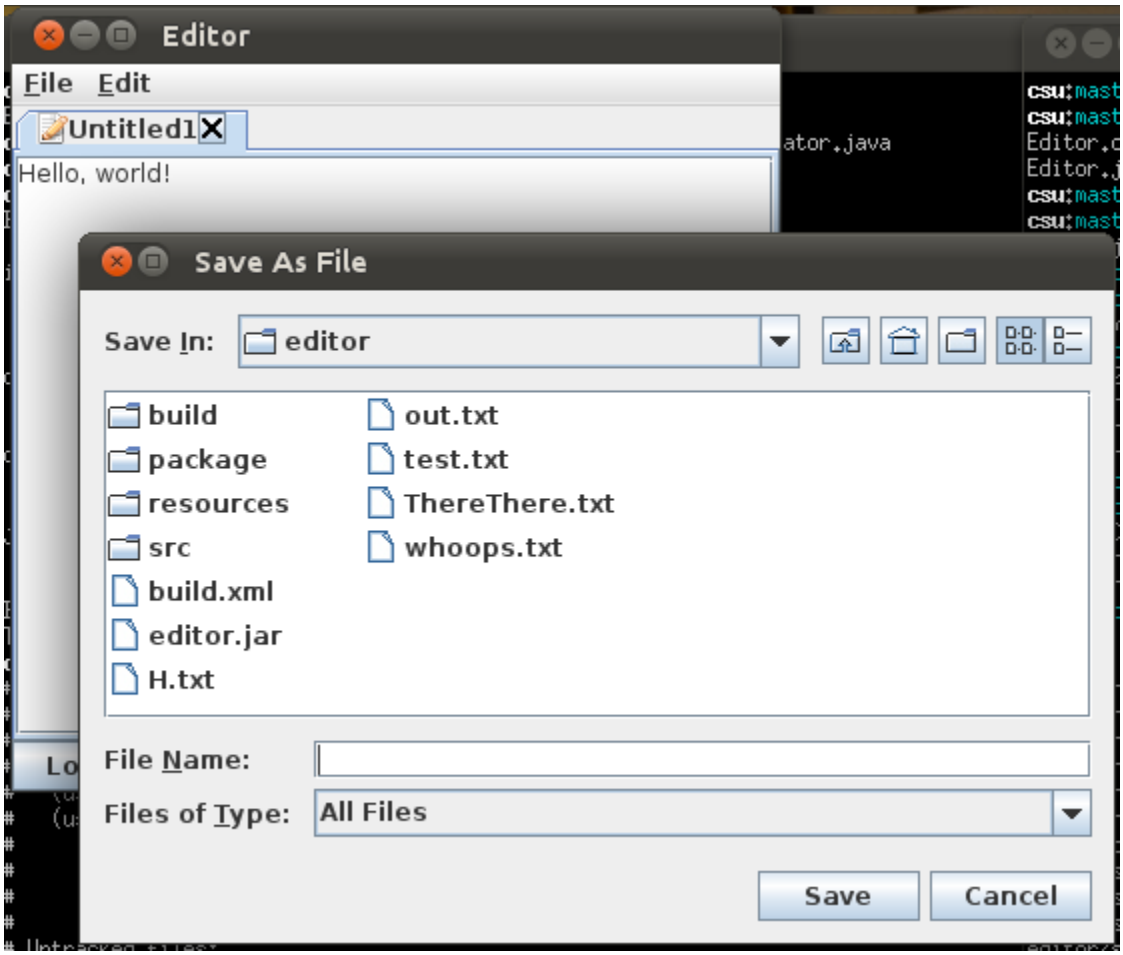
Edited file.



Open File Dialog.



Save File Dialog.



Editor with multiple tabs open.