```java
/**
 * The main file for the Editor Project.
 * @author ALL
 *
 */
public class MainFile
{
	MenuView menu;
	MainView view;
	Mediator mediator;
	InputHandler input;

	/**
	 * The constructor of the MainFile class.
	 */
	public MainFile()
	{
		mediator = new Mediator();
		input = new InputHandler(mediator);
		view = new MainView(input);
		mediator.setMainView(view);
	}

	public MainFile(String path)
	{
		this();

		String name = "";

		int lastPos = path.indexOf("\\");
		while(lastPos != -1){
			name = path.substring(lastPos + 1);
			lastPos = path.indexOf("\\", lastPos + 2);
		}

		mediator.openFile(name, path);
	}

	/**
	 * Main method.
	 * @param args
	 */
	public static void main(String args[]){
		if(args.length > 0 ){
			new MainFile(args[0]);
		}else{
			new MainFile();
		}
	}
}

import java.util.ArrayDeque;
import java.util.Deque;
import java.util.Stack;

/**
 * Holds all of the users text.
 * @author Adam, Braxton, Andrew
 *
 */
public class File{
	private Deque<Command> commandStack;
	private Stack<Command> redoStack;
	private String buffer;
	private int id;
	private int stackSize;
	private String location;
	private boolean isSaved;
	private boolean isFunctional;

	/**
	 * The constructor of the File class.
	 * @param int
	 */
	public File(int idNum){
		commandStack = new ArrayDeque<Command>();
		redoStack = new Stack<Command>();
		buffer = "";
		id = idNum;
		stackSize = 20;
		isSaved = false;
		isFunctional = false;
	}

	/**
	 * Overloaded Constructor of the File class.
 * @param b
 * @param idNum
 */
public File(String b,int idNum){
	this(idNum);
	buffer = b;
}

/**
 * Returns the id of the File.
 * @return
 */
public int getID(){
	return id;
}

/**
 * Returns the location of the File.
 * @return
 */
public String getPath(){
	return location;
}

/**
 * Returns the buffer of the File.
 * @return
 */
public String getBuffer(){
	return buffer;
}

/**
 * Boolean that checks to see if the File has been saved.
 * @return
 */
public boolean isSaved(){
	return isSaved;
}

/**
 * Sets whether the File was saved.
 * @param b
```

```
	 */
	public void setIsSaved(boolean b){
		isSaved = b;
	}

	/**
	 * Sets the location of the File.
	 * @param p
	 */
	public void setPath(String p){
		location = p;
	}

	/**
	 * Sets the buffer of the File.
	 * @param s
	 */
	public void setBuffer(String s){
		buffer = s;
	}

    /**
     * Saves the command to the file and then applies the command.
     * @param cmd
     */
	public void pushCommand(Command cmd){
		this.pushCommand(cmd, true);
	}

	public void pushCommand(Command cmd, boolean clear){
		if(cmd == null){
			System.out.println("cmd is null");

		}
		cmd.Apply(this);
		if(cmd.isUndoable){
			commandStack.addFirst(cmd);
			if(clear){
				redoStack.clear();
			}
		}

		if(commandStack.size() > stackSize){
				commandStack.removeLast();
			}
		}

		/**
		 * Removes and undo's the command.
		 */
		public void popCommand(){
			if(!commandStack.isEmpty()){
				commandStack.getFirst().Undo(this);
				redoStack.push(commandStack.getFirst());
				commandStack.pop();
			}
		}

		/**
		 * Redo's the recently undone command.
		 */
		public void redoCommand(){
			if(redoStack.size() == 0){return;}

			this.pushCommand(redoStack.pop(), false);
		}

		/**
		 * Well Formed Check.
		 * @return
		 */
		public boolean getIsFunctional(){
			return isFunctional;
		}

		/**
		 * Sets whether the File is well formed.
		 * @param b
		 */
		public void setIsFunctional(boolean b){
			isFunctional = b;
		}
}

import java.util.ArrayList;
import java.util.List;
```

```
/**
 * Allows the FileHandler to treat FileContent as a File object.
 * @author Braxton, Andrew, Adam
 *
 */
public class FileContent {
	private File activeFile;
	private List<File> fileList;

	/**
	 * Constructor of the FileContent class.
	 */
	public FileContent(){
		fileList = new ArrayList<File>();
		activeFile = null;
	}

	/**
	 * Changes the the active file
	 */
	public void changeFile(int id){
		for(int i = 0; i < fileList.size(); i++){
			if(fileList.get(i).getID() == id){
				activeFile = fileList.get(i);
				return;
			}
		}
	}

	/**
	 * Sends the parameter command to the active file
	 * @param cmd
	 */
	public void pushCommand(Command cmd){
		if(activeFile != null)
			activeFile.pushCommand(cmd);
	}

	/**
	 * Undoes the most recent command of the active file
	 */
	public void popCommand(){
		if(activeFile != null)
			activeFile.popCommand();
	}

	/**
	 * Redo's the recently undone command.
	 */
	public void redoCommand(){
		if(activeFile != null)
			activeFile.redoCommand();
	}

	/**
	 * Adds a new file to the file list.
	 * @param file
	 */
	public void addFile(File file){
		fileList.add(file);
	}

	/**
	 * Returns the active files location.
	 * @return
	 */
	public String getPath(){
		if(activeFile == null){
			return "";
		}

		return activeFile.getPath();
	}

	/**
	 * Returns the active files buffer.
	 * @return
	 */
	public String getBuffer(){
		if(activeFile == null){
			return "";
		}

		return activeFile.getBuffer();
	}
```

```
/**
 * Set's whether the active file has been saved.
 * @param b
 */
public void setIsSaved(boolean b){
        if(activeFile == null){
                return;
        }

        activeFile.setIsSaved(b);
}

/**
 * Return's whether the active file has been saved.
 * @return
 */
public boolean getIsSaved(){
        if(activeFile == null){
                return true;
        }

        return activeFile.isSaved();
}

/**
 * Returns the active files id.
 * @return
 */
public int getID(){
        if(activeFile == null){
                return -1;
        }

        return activeFile.getID();
}

/**
 * Sets the active files buffer.
 * @param s
 */
public void setBuffer(String s){
        if( s != null && activeFile != null)
```

```
			activeFile.setBuffer(s);
	}

	/**
	 * Returns the active file.
	 * @return
	 */
	public File getActiveFile(){
		return activeFile;
	}

	/**
	 * Sets whether the active file is well formed.
	 * @param b
	 */
	public void setIsFunctional(boolean b){
		if(activeFile == null){
			return;
		}

		activeFile.setIsFunctional(b);
	}

	/**
	 * Returns whether the active file is well formed.
	 * @return
	 */
	public boolean getIsFunctional(){
		if(activeFile == null){
			return false;
		}

		return activeFile.getIsFunctional();
	}

	/**
	 * Finds file by id. Returns File.
	 * @param id
	 * @return
	 */
	public File getFileByID(int id){
		for(File f : fileList){
			if(f.getID() == id){
				return f; //file is found, return the file
			}
		}
		return null;//File doesn't exist, return NULL
	}

	/**
	 * Removes file from list.
	 * @param file
	 */
	public void removeFile(File file){
		if(activeFile == null){
			return;
		}

		fileList.remove(file);
		if( activeFile == file ){
			activeFile = null;

			if(fileList.size() > 0){
				activeFile = fileList.get( fileList.size() - 1 ); //most recently added
file is now the active file
			}

		}
	}

	/**
	 * Sets location of active file.
	 * @param path
	 */
	public void setPath( String path){
		if(activeFile != null)
			activeFile.setPath(path);
	}
}

import java.util.ArrayList;
import java.util.List;
import java.util.Stack;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

/**
 * Handles the loading and saving of the file objects.
 * @author Adam, Andrew, Braxton
 *
 */
public class FileHandler {
	private FileContent fileContent;
	private Mediator mediator;
	private int fileNumbers;
	private FormatHelper formatHelper;

	/**
	 * The constructor of the FileHandler class.
	 * @param med
	 */
	public FileHandler(Mediator med){
		fileNumbers = 0;
		fileContent = new FileContent();
		mediator = med;
		formatHelper = new FormatHelper();
	}

	// ACCESSORS
	/**
	 * Sets whether the file is saved.
	 * @param b
	 */
	public void setIsSaved(boolean b){
		fileContent.setIsSaved(b);
	}

	/**
	 * Returns whether the file is well formed.
	 * @return
	 */
	public boolean getIsFunctional(){
		setIsFunctional();
		return fileContent.getIsFunctional();
	}
	/**
	 * Sets whether the file is well formed.
	 */
	public void setIsFunctional(){
		fileContent.setIsFunctional(wellFormed(fileContent.getActiveFile()));
	}

	/**
	 * Checks to see if the file can save.
	 * @return
	 */
	public boolean canSave(){
		return fileContent.getActiveFile() != null;
	}

	/**
	 * Saves the active file
	 */
	public boolean save(){
		FileWriter fw;
		BufferedWriter bw;

		if(fileContent.getPath() != null){
			try{
				fileContent.setBuffer(mediator.getMainViewText());

				fw = new FileWriter(fileContent.getPath());
				bw= new BufferedWriter(fw);
				bw.write(fileContent.getBuffer());
				bw.close();

				fileContent.setIsSaved(true);
			}

			catch (IOException e1){
				System.out.println("Error saving file '" + fileContent.getPath() +
"'");
				e1.printStackTrace();
			}

			return true;
		}
	else return false;
}

public String getBuffer(){
	return fileContent.getBuffer();
}
//END ACCESSORS

// COMMANDS
/**
 * Sends the parameter command to the active file
 * @param cmd
 */
public void pushCommand(Command cmd){
	fileContent.pushCommand(cmd);
	updateDisplay();
}

/**
 * Undoes the most recent command of the active file
 */
public void popCommand(){
	fileContent.popCommand();
	updateDisplay();
}

/**
 * Redo's the recently undone command.
 */
public void redoCommand(){
	fileContent.redoCommand();
	updateDisplay();
}
// END COMMANDS

/**
 * Sets the active file's buffer
 * @param s
 */
public void updateFileBuffer(String s){
	fileContent.setBuffer(s);
}
```

```
/**
 * Updates the Text box with the backends buffer.
 */
public void updateDisplay(){
        mediator.setTextAreaString(fileContent.getBuffer());
}

/**
 * Changes the active file.
 * @param id
 */
public void changeCurrentFile(int id){
        fileContent.changeFile(id);
}


/**
 * Prompts the user to save the file with a desired name and location
 */
public void saveAs(String path){
        FileWriter fw;
        BufferedWriter bw;

        try{
                fw = new FileWriter(path);
                bw= new BufferedWriter(fw);
                bw.write(fileContent.getBuffer());
                bw.close();

                //Update Name
                mediator.updateTabName(getNameFromPath(path));
                //Update file path
                fileContent.setPath(path);
        }

        catch (IOException e1){
                System.out.println("Error saving file '" + path + "'");
                e1.printStackTrace();
        }
}
```

```
/**
 * Creates new instance of a File.
 * @param name
 * @return
 */
public File createNewFile(String name){
        File newFile;
        newFile = new File("",fileNumbers);
        fileNumbers +=1;

        fileContent.addFile(newFile);
        fileContent.changeFile(newFile.getID());
        newFile.setIsFunctional(true);
        return newFile;
}

/**
 * Loads a file from the location specified in the parameter
 * @param loc
 * @return null
 */
public File load(String loc){
        File newFile;
        String line = "";
        List<String> newBuffer = new ArrayList<String>();
        FileReader fileReader;
        BufferedReader bufferedReader;

        try{
                fileReader = new FileReader(loc);
                bufferedReader = new BufferedReader(fileReader);
                newBuffer = new ArrayList<String>();

                // stores each line of the file in a list
                while((line = bufferedReader.readLine()) != null) {
            newBuffer.add(line);
        }

                bufferedReader.close();
        }

        catch (FileNotFoundException e) {
                System.out.println("Unable to open file '" + loc + "'");
```

```
				e.printStackTrace();
			}

			catch (IOException e) {
				System.out.println("Error reading file '" + loc + "'");
				e.printStackTrace();
			}

			// used to append the list of lines into a single line
			StringBuilder builder = new StringBuilder();
			for(String s:newBuffer){
				builder.append(s);
				builder.append("\n"); // without new line characters it is a 1 line string
			}

			// create the new file and increment unique file ID
			newFile = new File(builder.toString(),fileNumbers);
			newFile.setPath(loc);
			fileNumbers +=1;

			fileContent.addFile(newFile);
			fileContent.changeFile(newFile.getID());
			setIsFunctional();
			return newFile;
		}

		/**
		 * Closes the file and removes it from the list.
		 * @param id
		 * @return
		 */
		public boolean close(int id){
			//If the file is unsaved, then prompt user to continue
			File file = fileContent.getFileByID(id);

			if(file.isSaved()){
				fileContent.removeFile(file); //removes the file from the list
				return true; //tells tabView to remove Tab
			}else{
				if( mediator.promptManager.displayBool(
						"The file you are attempting to close is not saved. Do you
wish to proceed?")){
					fileContent.removeFile(file); //removes the file from the list
```

```
                        return true; //tells tabView to remove Tab
                }else{
                        return false;
                }
        }
}

public String getTagLayout(){
        return formatHelper.formatTabbedString(fileContent.getBuffer());
}

//TODO move wellformed to formathelper
/**
 * Checks to see if the text in the current file is valid HTML
 * @return boolean
 */
public boolean wellFormed(File file){
        if(file == null){
                return false;
        }

        List<String> leftOverTags = new ArrayList<String>();
        Stack<String> tagStack = new Stack<String>();
        String allText = file.getBuffer(); // the buffer from the file

        int start = allText.indexOf('<');
        int end = allText.indexOf('>');
        String tag = "";

        if((start == -1 || end == -1) || start > end){
                if(start == -1 && end == -1){
                        //if there are no html tags in the file
                        return true;
                }

                //missing a > or <
                notifyIllformed();
                return false;
        }

        while((start != -1 && end != -1)){
                if(start >= end){
                        notifyIllformed();
                return false;
            }

            tag = allText.substring(start + 1, end); // changes tag from <b> to b>
            //if it is an open tag
            if(tag.charAt(0) != '/'){
                if(tag.charAt(0) != '!'){ //ignores the opening <!HTMLDOC> tag
                    if(checksSelfClose(tag)){
                        leftOverTags.add(tag);
                    }

                    else if(tag.contains("img src")){ //for <img src...> tags
                        tag = "img";
                        tagStack.push(tag);
                    }

                    else if(tag.contains("a href")){ //for <a href...> tags
                        tag = "a";
                        tagStack.push(tag);
                    }

                    else{
                        tagStack.push(tag);
                    }
                }
            }

            //if it is a close tag
            else if(tag.charAt(0) == '/'){
                tag = tag.substring(tag.indexOf('/') + 1); //find the actual tag by
removing the close character
                if(tagStack.peek().equals(tag)){ //check to see if the most recent
tag is the open tag for this close tag
                    tagStack.pop(); //if it is remove the open tag
                }

                else{
                    //a mismatched close tag has been found
                    notifyIllformed();
                    return false;
                }
            }
```

```
				//update the text to go through
				allText = allText.substring(end+1);
				start = allText.indexOf('<'); //find the next tag start and end
				end = allText.indexOf('>');
			}

			//if there are leftover tags that arent closed, and they aren't self closing tags
			if(tagStack.size() != 0 && tagStack.size() != leftOverTags.size()){
				notifyIllformed();
				return false;
			}

			else return true;
		}

		//PRIVATE METHODS
		/**
		 * Returns the name from the path of the file.
		 * @param path
		 * @return
		 */
		private String getNameFromPath( String path ){
			String name = "";

			int lastPos = path.indexOf("\\");
			while(lastPos != -1){
				name = path.substring(lastPos + 1);
				lastPos = path.indexOf("\\", lastPos + 2);
			}
			return name;
		}

		/**
		 * Fires a prompt, informing the user that their HTML is not well formed.
		 */
		private void notifyIllformed(){
			mediator.promptManager.displayMessage("Your file contains illformed HTML,
	some functionality may be disabled till this is corrected");
		}

		/**
		 * Checks to see if a tag is valid despite not following
		 * the standard tag format
	 * @param tag
	 * @return
	 */
	private boolean checksSelfClose(String tag){
		String[] selfClosing = {"meta", "link", "input", "tr"};

		for(int i = 0; i < selfClosing.length; i++){
			if(selfClosing[i].equals(tag)){
				return true;
			}
		}

		return false;
	}


} // END FILEHANDLER

/**
 * Abstract class for the Command classes.
 * @author Braxton.
 *
 */
public abstract class Command {

	public boolean isUndoable;
	protected String text;
	protected String buffer;
	protected int start;
	protected int end;

	public abstract void Apply(File file);

	public abstract void Undo(File file);
}

/**
 * Builds all of the concrete command classes.
 * @author Braxton
 *
 */
public class CommandBuilder{
	private Mediator mediator;
```

```
public CommandBuilder(Mediator _m){
	mediator = _m;
}

/**
 * Handles creation of commands
 * @param text
 * @param start
 * @param end
 * @param type
 * @return Command
 */
public Command CreateCommand(String text, int start, int end, String type){
	Command cmd;

	int temp = start;

	if(start > end){
		start = end;
		end = temp;
	}

	if(type == "Additive"){
		cmd = new AdditiveCommand(text,start,end);
	}

	else if(type == "Subtractive"){
		cmd = new SubtractiveCommand(text,start,end);
	}

	else if(type == "tag"){
		cmd = new InsertTagCommand(text, start, end);
	}

	else if(type == "link"){
		String s = mediator.promptManager.displayLines1("Enter the url:");
		if(s != ""){
			cmd = new InsertLinkCommand(s, start, end);
		}
		else cmd = new ErrorCommand(); //do nothing
	}
```

```
		else if(type == "list"){

			String s = mediator.promptManager.displayLines1("Enter the number of
list elements:");
			if(s != ""){
				try{
					int i = Integer.parseInt(s);
					cmd = new InsertListCommand(text, start, i);
				}
				catch(Exception e){
					cmd = new ErrorCommand(); //do nothing
				}
			}

			else cmd = new ErrorCommand(); //this intentionally does nothing, cmd
must be returned
		}

		else if(type == "table"){
			String[] userInput = new String[2];
			userInput = mediator.promptManager.displayLines2("Number of rows:",
"Number of columns:");

			try{
				int i = Integer.parseInt(userInput[0]);
				int j = Integer.parseInt(userInput[1]);
				cmd = new InsertTableCommand(start, i, j);
			}

			catch(Exception e){
				cmd = new ErrorCommand(); //do nothing
			}
		}

		else if(type == "img"){
			String s = mediator.promptManager.displayLines1("Enter the source
path:");
			if(s != ""){
				cmd = new InsertImageCommand(s, start, end);
			}
			else cmd = new ErrorCommand(); //do nothing
		}
		else{
			cmd = null;
		}

		return cmd;
	}
}

/**
 * Adds text to the given position.
 * @author Braxton
 *
 */
public class AdditiveCommand extends Command{

	public AdditiveCommand(String textString, int startPosition, int endPosition){
		text = textString;
		start = startPosition;
		end = endPosition;

		if(end == start){
			end = start + text.length();
		}

		this.isUndoable = false;
	}

	/**
	 * Updates a file's text by replacing
	 * it with the text + a desired substring
	 */
	public void Apply(File file){
		String b = file.getBuffer();

		if(text.equals(b)){
			return; //buffers are the same
		}else{
			isUndoable = true;
			buffer = b;
			file.setBuffer(text);
		}
	}
	/**
	 * Undoes the addition of text from a file
	 */
	public void Undo(File file){
		file.setBuffer(buffer);
	}
}

/**
 * Command that deletes text.
 * @author Braxton
 *
 */
public class SubtractiveCommand extends Command{

	/**
	 * The constructor for the SubtractiveCommand class.
	 * @param textString
	 * @param startPosition
	 * @param endPosition
	 */
	public SubtractiveCommand(String textString, int startPosition, int endPosition){
		text = textString;
		start = startPosition;
		end = endPosition;
		isUndoable = true;
	}

	/**
	 * Updates a file's text by replacing
	 * it with the text - a desired substring
	 */
	public void Apply(File file){
		buffer = file.getBuffer();
		file.setBuffer(text);
	}

	/**
	 * Undoes the removal of text from a file
	 */
	public void Undo(File file){
		file.setBuffer(buffer);
	}
```

```
}

/**
 * Error Command that is send when a inserting command is canceled.
 * @author Andrew
 *
 */
public class ErrorCommand extends Command {

	public ErrorCommand(){
		this.isUndoable = false;
	}

	/**
	 * Is not used in this class.
	 */
	public void Apply(File file) {

	}

	/**
	 * Is not used in this class.
	 */
	public void Undo(File file) {

	}
}

/**
 * Command that is sent when an image is inserted.
 * @author Adam
 *
 */
public class InsertImageCommand extends Command {

	/**
	 * Constructor for the image inserting command
	 * @param src
	 * @param startPosition
	 * @param endPosition
	 */
	public InsertImageCommand(String src, int startPosition, int endPosition){
		text = src;
```

```
			start = startPosition;
			end = endPosition;
			isUndoable = true;
		}

		/**
		 * Updates a file's text by replacing a string with the string plus a substring.
		 * @param file
		 */
		@Override
		public void Apply(File file) {
			buffer = file.getBuffer();
			String newBuffer = buffer.substring(0,start) + "<img src=" + "\"" + text + "\"" +
">" + buffer.substring(start,end) + "</img>"+ buffer.substring(end);
			file.setBuffer(newBuffer);
		}

		/**
		 * Undoes the addition of text.
		 * @param file
		 */
		@Override
		public void Undo(File file) {
			file.setBuffer(buffer);
		}
}

/**
 * Command for inserting a link.
 * @author Adam
 */
public class InsertLinkCommand extends Command {

		/**
		 * Constructor for link inserting command
		 * @param url
		 * @param startPosition
		 * @param endPosition
		 */
		public InsertLinkCommand(String url, int startPosition, int endPosition){
			text = url;
			start = startPosition;
			end = endPosition;
			isUndoable = true;
		}

		/**
		 * Updates a file's text by replacing
		 * it with the text + a desired substring
		 * @param file
		 */
		public void Apply(File file){
			buffer = file.getBuffer();
			String newBuffer = buffer.substring(0,start) + "<a href=" + "\"" + text + "\"" + ">"
	+ buffer.substring(start,end) + "</a>"+ buffer.substring(end);
			file.setBuffer(newBuffer);
		}

		/**
		 * Undoes the addition of text from a file
		 * @param file
		 */
		public void Undo(File file){
			file.setBuffer(buffer);
		}
	}

	/**
	 * Command for inserting a List
	 * @author Adam
	 */
	public class InsertListCommand extends Command {

		/**
		 * Constructor for list inserting command
		 * @param file
		 */
		public InsertListCommand(String tag, int start, int numRows){
			text = "";
			text += '<' + tag + '>';

			for(int i = 0; i < numRows; i++){
				text += '\n';

				if(tag == "dl"){
					text += "<dt> </dt>" + '\n' + "<dd> </dd>";
```

```
                        }else{
                                text += "<li> </li>";
                        }
                }

                text += '\n' + "</" + tag + ">";

                this.start = start;
                isUndoable = true;
        }

        /**
         * Updates a file's text by replacing
         * it with the text + a desired substring
         * @param file
         */
        public void Apply(File file) {
                buffer = file.getBuffer();

           String newBuffer = buffer.substring(0,start) + text + buffer.substring(start);

                file.setBuffer( newBuffer );
        }

        /**
         * Undoes the addition of text from a file
         * @param file
         */
        public void Undo(File file) {
                file.setBuffer(buffer);
        }

}

/**
 * Command for inserting a table.
 * @author Braxton
 */
public class InsertTableCommand extends Command {

        /**
         * Constructor for table inserting command
         * @param startPosition
```

```
	 * @param _rows
	 * @param _cols
	 */
	public InsertTableCommand(int startPosition, int _rows, int _cols){
		start = startPosition;

		//construct table
		text = "<table>";
		for(int r = 0; r < _rows; r++){
			text += "\n <tr>";
			for(int c = 0; c < _cols; c++){
				text += "\n <td> </td>";
			}
			text += '\n' + "</tr>";
		}
		text += "\n </table>";
		isUndoable = true;
	}


	/**
	 * Updates a file's text by replacing
	 * @param file
	 */
	public void Apply(File file){
		buffer = file.getBuffer();
		String newBuffer = buffer.substring(0,start) + text + buffer.substring(start);
		file.setBuffer(newBuffer);
	}

	/**
	 * Undoes the addition of text from a file
	 * @param file
	 */
	public void Undo(File file){
		file.setBuffer(buffer);
	}
}

/**
 * Command for inserting tags
 * @author Adam
 */
```

```
public class InsertTagCommand extends Command{

	/**
	 * Constructor for tag insertion command
	 * @param textString
	 * @param startPosition
	 * @param endPosition
	 */
	public InsertTagCommand(String textString, int startPosition, int endPosition){
		text = textString;
		start = startPosition;
		end = endPosition;
		isUndoable = true;
	}

	/**
	 * Updates a file's text by replacing
	 * it with the text + a desired substring
	 * @param file
	 */
	public void Apply(File file){
		buffer = file.getBuffer();

		int temp = start;
		if(start > end){
			start = end;
			end = temp;
		}

		String newBuffer = buffer.substring(0,start) + "<"+ text + ">" +
buffer.substring(start,end) + "</" + text + ">"+ buffer.substring(end);
		file.setBuffer(newBuffer);
	}

	/**
	 * Undoes the addition of text from a file
	 * @param file
	 */
	public void Undo(File file){
		file.setBuffer(buffer);
	}
}
```

```
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;

/**
 * LinkedView deals with displaying the URL tags from the editors buffer.
 * @author Adam, Andrew
 *
 */
@SuppressWarnings("serial")
public class LinkedView extends JFrame {
	JTextArea linkedViewList;
	JPanel contentPane;
	JButton button;
	LinkedViewStrategy strategy;
	String fileBuffer;
	MainView mainView;

	/**
	 * Creates and displays the LinkedView JFrame.
	 * @param m
	 * @param strategy
	 * @param text
	 */
	public LinkedView(MainView m, LinkedViewStrategy strategy,String text){
		this.strategy = strategy;
		fileBuffer = text;
		mainView = m;

		this.setMinimumSize(new Dimension(300,450));
		this.setTitle(strategy.getName());

		this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);

        contentPane = new JPanel();
        contentPane.setLayout(new BorderLayout());
        button = new JButton("Refresh");

        button.addActionListener(new ActionListener(){
           @Override
           public void actionPerformed(ActionEvent e){
                     mainView.getInputHandler().updateLinkedView();
           }
        });

        linkedViewList = new JTextArea();
        linkedViewList.setPreferredSize(new Dimension(this.getWidth(), this.getHeight() -
button.getHeight()));
        linkedViewList.setEditable(false);

        contentPane.add(linkedViewList, BorderLayout.NORTH);
        contentPane.add(button, BorderLayout.SOUTH);

        List<String> bufferList = strategy.parse(fileBuffer);
        List<Integer> intList = strategy.numOccur();

        for(int i = 0; i < bufferList.size(); i++){
           //for alphabetical
           if(intList != null){
                     linkedViewList.append("url: " + bufferList.get(i) + " count: " + intList.get(i) + '\n');
           }

           //for appearance
           else{
                     linkedViewList.append("url: " + bufferList.get(i) + '\n');
           }
        }

        this.setContentPane(contentPane);
        this.pack();
        this.setLocationRelativeTo(mainView);
        this.setVisible(true);
           }

           /**
            * Method that updates the LinkView's list of URL's.
            * @param newBuffer
            */
```

```
	public void updateLinkList(String newBuffer){
		fileBuffer = newBuffer;
		linkedViewList.setText("");

		List<String> bufferList = strategy.parse(fileBuffer);
	    List<Integer> intList = strategy.numOccur();

	    for(int i = 0; i < bufferList.size(); i++){
		//for alphabetical
		if(intList != null){
			linkedViewList.append("url: " + bufferList.get(i) + " count: " + intList.get(i)
+ '\n');
	        }

	        //for appearance
	        else{
		   linkedViewList.append("url: " + bufferList.get(i) + '\n');
	        }
	    }
	}
}

import java.util.List;

/**
 * Abtract Strategy for the LinkedView class.
 * @author Adam
 *
 */
public interface LinkedViewStrategy {

	public List<String> parse(String buffer);

	public List<Integer> numOccur();

	public String getName();
}

import java.util.ArrayList;
import java.util.List;

/**
```

```
 * Implementation of the LinkedViewStrategy. Sorts the URL list by alphabetical order. Lists
the number of occurrences.
 * @author Adam, Andrew
 *
 */
public class SortByAlpha implements LinkedViewStrategy{
	private String[] splitText;
	private List<String> urlList;
	private List<Integer> urlOccurance;
	private String name;

	/**
	 * Constructor of the SortByAlpha class.
	 */
	public SortByAlpha(){
		urlList = new ArrayList<String>();
		urlOccurance = new ArrayList<Integer>();
		this.name = "Alphabetical Sort";
	}

	/**
	 *Returns the number of occurrences that each URL appear.
	 */
	public List<Integer> numOccur(){
		return urlOccurance;
	}

	/**
	 * Returns the name of the Strategy type.
	 */
	public String getName(){
		return name;
	}

	/**
	 * Looks at the buffer and stripes out the URL tags.
	 * @return List<String>
	 */
	public List<String> parse(String buffer) {
		urlList = new ArrayList<String>();
		urlOccurance = new ArrayList<Integer>();

		splitText = buffer.split("[\n]+");
		String[] tempList;

		for(int i = 0; i < splitText.length; i++){ //goes through each line of code
			tempList = splitText[i].split("<a href=+");

			for(int j = 0; j < tempList.length; j++){
				if(tempList[j].startsWith("\"")){
					if(urlList.contains(tempList[j])){
						int element = urlList.indexOf(tempList[j]);
						urlOccurance.set(element,
urlOccurance.get(element) + 1);
					}

					else{
						urlList.add(tempList[j]);
						urlOccurance.add(1);
					}
				}
			}
		}

		int end = 0;

		for(int i = 0; i < urlList.size(); i++){
			end = urlList.get(i).indexOf(">");
			if(end > 0){
				urlList.set(i, urlList.get(i).substring(0,end));
			}
		}

		sort();
		return urlList;
	}

	/**
	 * Orders the URL list in alphabetical order.
	 */
	private void sort(){
		int j;
		boolean flag = true;
		String temp1;
		int temp2;
```

```
			while(flag){
				flag = false;
				for(j = urlList.size() - 1; j > 0; j--){
					if(urlList.get(j).compareTo(urlList.get(j - 1)) < 0){
						temp1 = urlList.get(j);
						urlList.set(j, urlList.get(j - 1));
						urlList.set(j - 1, temp1);

						temp2 = urlOccurance.get(j);
						urlOccurance.set(j, urlOccurance.get(j - 1));
						urlOccurance.set(j - 1, temp2);

						flag = true;
					}
				}
			}
		}
}

import java.util.ArrayList;
import java.util.List;

/**
 *
 * Implementation of the LinkedViewStrategy. Sorts the URL list by order of appearance.
 * @author Adam, Andrew
 *
 */
public class SortByAppear implements LinkedViewStrategy{
		private String[] splitText;
		private List<String> urlList;
		private String name;

		/**
		 * Constructor of the SortByAppear class.
		 */
		public SortByAppear()
		{
			urlList = new ArrayList<String>();
			this.name = "Appearance Sort";
		}

		/**
 * Returns null beacause this method is only used in SortByAlpha.
 */
public List<Integer> numOccur() {
	return null;
}

/**
 * Returns the name of the Strategy type.
 */
public String getName(){
	return name;
}

/**
 * Looks at the buffer and stripes out the URL tags.
 * @return List<String>
 */
public List<String> parse(String buffer){
	urlList = new ArrayList<String>();

	splitText = buffer.split("[\n]+");
	String[] tempList;

	for(int i = 0; i < splitText.length; i++){
		tempList = splitText[i].split("<a href=+");

		for(int j = 0; j < tempList.length; j++){
			if(tempList[j].startsWith("\"")){
				urlList.add(tempList[j]);
			}
		}
	}

	int end;
	for(int i = 0; i < urlList.size(); i++){
		end = urlList.get(i).indexOf(">");
		if(end > 0){
			urlList.set(i, urlList.get(i).substring(0,end));
		}
	}

	return urlList;
}
```

```
}

import java.awt.GridLayout;
import java.awt.Label;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

/**
 * Creates the prompts that ask for the user's input.
 * @author Andrew, Braxton, Arron
 *
 */
public class PromptManager{

        String message;
        MainView mainView;
        JFrame parent;
    boolean returnType;


        /**
         * The constructor for the PromptManager.
         * @param m
         */
        public PromptManager(MainView m){
                message = "";
                mainView = m;
        }

        /**
         * Displays a message to the user
         * @param message
         */
        public void displayMessage(String m){
                parent = new JFrame();

                JOptionPane pane = new JOptionPane();
        pane.setMessage(m);

        JButton ok = new JButton("Ok");
        Object[] options = {ok};
        pane.setOptions(options);

        ok.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e){
                    parent.dispose();
          }
        });

        parent.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        parent.add(pane);
        parent.pack();
        parent.setLocationRelativeTo(null);
                    parent.setVisible(true);
          }

          /**
           * Displays a dialog for two input custom prompt
           * Shows message1, text box, message2, text box
           * @param message1, message2
           */
          public String[] displayLines2(String m1, String m2){
                    String[] textFields = new String[2];
                    String[] failed = {"",""};
                    JPanel panel = new JPanel(new GridLayout(2,2));
                    Label l1 = new Label(m1);
                    Label l2 = new Label(m2);
                    Object[] options = {"Ok"};
                    JTextField tF1 = new JTextField();
                    JTextField tF2 = new JTextField();
                    panel.add(l1);
                    panel.add(tF1);
                    panel.add(l2);
                    panel.add(tF2);

                    int result = JOptionPane.showOptionDialog(null, panel, message,
JOptionPane.OK_CANCEL_OPTION,
                                    JOptionPane.QUESTION_MESSAGE, null, options, null);
                    if(result == JOptionPane.OK_OPTION){
                              textFields[0] = tF1.getText();
```

```
                textFields[1] = tF2.getText();
                return textFields;
        }
        else return failed;
}

/**
 * Displays a dialog for one input custom prompt
 * Shows message, text box
 * @param message
 */
public String displayLines1(String m){
        JPanel panel = new JPanel(new GridLayout(2,1));
        Label label = new Label(m);
        Object[] options = {"OK", "Cancel"};
        JTextField textField = new JTextField();
        panel.add(label);
        panel.add(textField);

        int result = JOptionPane.showOptionDialog(
                                        null,
                                        panel,
                                        "Input",
                                        JOptionPane.OK_CANCEL_OPTION,
                                        JOptionPane.QUESTION_MESSAGE,
                                        null,
                                        options,
                                        null
                                );

        if(result == JOptionPane.OK_OPTION){
                return textField.getText();
        }
        else return "";
}

/**
 * Displays dialog with custom prompt
 * @param message
 */
public boolean displayBool(String message){
        return JOptionPane.showConfirmDialog(
                                new JFrame(),
                                        message,
                                        "Confirm",
                                        JOptionPane.YES_NO_OPTION
                                ) == 0;
        }

        /**
         * Displays dialog with custom prompt
         * @param message
         */
        public int displayChoice(String message, String option1, String option2){
                Object[] options = {option1, option2};

                //create dialog box.  Store user response
                return JOptionPane.showOptionDialog(
                                        new JFrame(),
                                message,
                                "Confirm",
                                JOptionPane.YES_NO_OPTION,
                                JOptionPane.QUESTION_MESSAGE,
                                null,
                                options,
                                options[0]
                                );
        }
}

public class Mediator{

        private CommandBuilder builder;
        private FileHandler fileHandler;
        private MainView mainView;
        public PromptManager promptManager;

        public Mediator(){
                builder = new CommandBuilder(this);
                fileHandler = new FileHandler(this);
        }

        /**
         * Sets the mainView variable stored in Mediator
         * @param m
         */
```

```
	public void setMainView(MainView m){
		mainView = m;
		promptManager = new PromptManager(mainView);
	}

	/**
	 * Tells the builder to create a command and then pushes it to fileHandler.
	 */
	public void pushCommand(String text, String type){
		int cursorStart = mainView.getCursorStart();
		int cursorEnd = mainView.getCursorEnd();

		if(type == "update"){

fileHandler.pushCommand(builder.CreateCommand(mainView.getText(), 0, text.length(),
"Additive"));
		}

		else if(type == "tagLayout"){

fileHandler.pushCommand(builder.CreateCommand(mainView.getText(), 0, text.length(),
"Additive"));
			text = fileHandler.getTagLayout();
			fileHandler.pushCommand(builder.CreateCommand(text, 0,
text.length(), "Additive"));
			return;
		}

		else if(type == "Subtractive"){

fileHandler.pushCommand(builder.CreateCommand(mainView.getText(), 0, text.length(),
"Subtractive"));

		}

		else{
			//update display

fileHandler.pushCommand(builder.CreateCommand(mainView.getText(), 0, text.length(),
"Additive"));
			fileHandler.pushCommand(builder.CreateCommand(text, cursorStart ,
cursorEnd, type));
```

```
			if(type == "link"){
				updateLinkedView();
			}
		}

		mainView.setCursorStart(cursorStart);
	}

	/**
	 * Removes the most recent command
	 */
	public void popCommand(){
		int cursorStart = mainView.getCursorStart();
		String mt = mainView.getText() ;
		String ft = fileHandler.getBuffer();
		if(!mt.equals(ft)){
			//update the backend before undoing
			pushCommand("", "update");
		}
		fileHandler.popCommand();
		mainView.setCursorStart(cursorStart);
	}

	/**
	 * Reapplies the most recently removed command
	 */
	public void redoCommand(){
		int cursorStart = mainView.getCursorStart();
		fileHandler.redoCommand();
		mainView.setCursorStart(cursorStart);
	}

	/**
	 * Sets the current file's text box's text to the string parameter
	 * @param s
	 */
	public void setTextAreaString(String s){
		//TODO eval removing this
		mainView.setText(s);
	}

	/**
	 * Returns the text from the text box of the current file
 * @return
 */
public String getMainViewText(){
	//TODO eval why this would be needed
	return mainView.getText();
}

public boolean canSave(){
	return fileHandler.canSave();
}

public boolean save(){
	return fileHandler.save();
}

public void saveAs(String path){
	fileHandler.saveAs(path);
}

public void quit(){
	mainView.quit();
}

public void openFile(String name, String path){
	File file = fileHandler.load(path);
	mainView.addTab(name, file.getID());
	setTextAreaString(file.getBuffer());
}

public void createNewFile(String name){
	File file = fileHandler.createNewFile(name);
	mainView.addTab(name, file.getID());
	setTextAreaString("");
}

public void changeCurrentFile(int id){
	fileHandler.changeCurrentFile(id);
}

public void setIsSaved(boolean b){
	fileHandler.setIsSaved(b);
}
```

```
	public void createNewLinkedView(){
		int strategy;
		strategy = promptManager.displayChoice("Please select a formatted 
view.","Alphabetical","Appearance");
		if(strategy > -1){
			mainView.newLinkedView(strategy);
		}
	}

	public void updateFileBuffer(){
		//TODO eval why this exists.
		pushCommand(getMainViewText(), "update");
	}

	public boolean getIsFunctional(){
		return fileHandler.getIsFunctional();
	}

	public void setIsFunctional(){
		fileHandler.setIsFunctional();
	}

	public boolean closeTab(int id){
		return fileHandler.close(id);
	}

	public void updateTabName(String name){
		mainView.updateFileName(name);
	}

	public void updateLinkedView(){
		if(mainView.linkedView != null){
			updateFileBuffer();
			mainView.linkedView.updateLinkList(getMainViewText());
		}
	}

	public void toggleWordWrap(){
		mainView.toggleWordWrap();
	}
}

import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;

public class InputHandler {

	private Mediator mediator;
	final JFileChooser fc;

	public InputHandler(Mediator m){
		mediator = m;
		fc = new JFileChooser();
		fc.setFileFilter(new FileNameExtensionFilter("HTML", "html"));
	}

	/**
	 * Handles the events fired from a button being pressed
	 * @param txt
	 */
	public void buttonViewInput(String tag){
		//TODO confirm update condition in this if statement is correct
		if(!tag.equals("Subtractive") && !tag.equals("update") &&
!mediator.getIsFunctional()){
			return;
		}

		String type = "tag";

		switch(tag){
			case "a": type = "link";break;
			case "ol": type = "list";break;
			case "ul": type = "list";break;
			case "dl": type = "list";break;
			case "table": type = "table";break;
			case "img": type = "img";break;
			case "update": type =tag; break;
			case "Subtractive": type=tag; break;
		}

		if(mediator.getMainViewText() != null){
			mediator.pushCommand(tag, type);
		}
	}

	/**
 * Handles the events fired from a menu selection
 * @param txt
 */
public void menuViewInput(String txt){
	switch(txt){
		case "New":
			mediator.createNewFile("new");
			break;

		case "Save":
			if(mediator.canSave()){
				if(mediator.save() == true){
					mediator.setIsSaved(true);
				}
				else menuViewInput("Save As...");
			}
			break;

		case "Save As...":
			if(mediator.canSave()){
				int returnVal = fc.showSaveDialog(fc);

				if(returnVal == JFileChooser.APPROVE_OPTION){
					java.io.File file = fc.getSelectedFile();
					String location = file.getPath().toString();

					mediator.saveAs(location);
					mediator.setIsSaved(true);
				}
			}
			break;

		case "Open File...":
			int returnVal = fc.showOpenDialog(fc);

			if(returnVal == JFileChooser.APPROVE_OPTION){
				java.io.File file = fc.getSelectedFile();
				mediator.openFile(file.getName(), file.getPath());
			}

			else System.out.println("Error opening file");
			break;
}

                case "Exit":
                        mediator.quit();
                        break;

                case "Undo":
                        mediator.popCommand();
                        break;

                case "Redo":
                        mediator.redoCommand();
                        break;

                case "Word Wrap":
                        if(mediator.getIsFunctional()){
                                mediator.toggleWordWrap();
                        }
                        break;

                case "Linked view":
                        if(mediator.getIsFunctional()){
                                mediator.createNewLinkedView();
                        }
                        break;

                case "Preview image":
                        if(mediator.getIsFunctional()){
                                new ImagePreviewer(mediator.getMainViewText());
                        }
                        break;

                case "Tag layout":
                        if(mediator.getIsFunctional()){
                                mediator.pushCommand("", "tagLayout");
                        }
                        break;
        }
}

public void changeCurrentFile(int id){
        mediator.changeCurrentFile(id);
}

public void updateFileBuffer(){
		mediator.updateFileBuffer();
	}

	public void setIsSaved(boolean b){
		mediator.setIsSaved(b);
	}

	public void quit(){
		mediator.quit();
	}

	public boolean closeTab(int id){
		return mediator.closeTab(id);
	}

	public void updateLinkedView(){
		mediator.updateLinkedView();
	}
}

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.util.ArrayList;
import java.util.List;

import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

/**
 * ImagePreviewer reads and displays the given image location.
 * @author Adam, and Andrew
 *
 */
```

```
public class ImagePreviewer extends JFrame{

    BufferedImage image;
    String path;
    JComboBox<String> comboBox;
    JPanel buttonPanel;
    JLabel label;

    /**
     * The constructor of the ImagePreviewer class.
     * @param buffer
     */
    public ImagePreviewer(String buffer){

        path = "";
        getImagePath(buffer);
    }

    /**
     * Prompts the user for the correct image that they want to preview.
     * @param buffer
     * @return String
     */
    public String getImagePath(String buffer){
        this.setMinimumSize(new Dimension(300, 100));

        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());

        label = new JLabel("Select the image to preview: ");

        List<String> imgList = parse(buffer);

        comboBox = new JComboBox<String>();

        for(int i = 0; i < imgList.size(); i++){
            comboBox.addItem(imgList.get(i));
        }

        buttonPanel = new JPanel();
        buttonPanel.setLayout(new GridLayout(1,2));

        JButton yesButton = new JButton("Ok");
    JButton noButton = new JButton("Cancel");

    yesButton.addActionListener(new ActionListener(){
    @Override
    public void actionPerformed(ActionEvent e){
            path = comboBox.getSelectedItem().toString();
            System.out.println(path);
            try{
                    displayImage(path);
            }

            catch(Exception ex){
                    ex.printStackTrace();
            }

            preview();
    }
  });

    noButton.addActionListener(new ActionListener(){
    @Override
    public void actionPerformed(ActionEvent e){
            endPreviewer();
    }
  });

    buttonPanel.add(yesButton);
    buttonPanel.add(noButton);

    label.setVisible(true);
    comboBox.setVisible(true);
    buttonPanel.setVisible(true);

    panel.add(label, BorderLayout.NORTH);
    panel.add(comboBox, BorderLayout.CENTER);
    panel.add(buttonPanel, BorderLayout.SOUTH);

    this.add(panel);

    this.pack();
    this.setContentPane(panel);
    this.setLocationRelativeTo(null);
    this.setVisible(true);
```

```
        return path;
    }

    /**
     * Sets all of the non image portions of the display to non visable.
     */
    public void preview()
    {
        label.setVisible(false);
        buttonPanel.setVisible(false);
        comboBox.setVisible(false);
    }

    /**
     * Reads in and displays the selected image.
     * @param path
     */
    public void displayImage(String path){
         try{
               String tempPath = "";
               tempPath = path.replaceAll("\\\\", "\\\\\\\\");
               tempPath = tempPath.replaceAll("\"", "");
               image = null;

               System.out.println(new File(tempPath).exists());
               System.out.println(tempPath);

               image = ImageIO.read(new File(tempPath));

               ImageIcon imageIcon = new ImageIcon(image);
         JLabel jLabel = new JLabel();
         this.setName("Image Preview");
         jLabel.setIcon(imageIcon);
         jLabel.setName("Image Preview");
         this.getContentPane().add(jLabel, BorderLayout.CENTER);

         this.pack();
         this.setLocationRelativeTo(null);
         this.setVisible(true);

         this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
       }
```

```
            catch (Exception e){
              e.printStackTrace();
          }
      }

      /**
       * Reads in the buffer and stripes the image paths from it.
       * @param buffer
       * @return List<String>
       */
      public List<String> parse(String buffer){
                    List<String> imgList = new ArrayList<String>();
                    String[] splitText;

                    splitText = buffer.split("[\n]+");
                    String[] tempList;

                    for(int i = 0; i < splitText.length; i++){
                              tempList = splitText[i].split("<img src=+");

                              for(int j = 0; j < tempList.length; j++){
                                        if(tempList[j].startsWith("\"")){
                                                  imgList.add(tempList[j]);
                                        }
                              }
                    }

                    int end;
                    for(int i = 0; i < imgList.size(); i++){
                              end = imgList.get(i).indexOf(">");
                              if(end > 0){
                                        imgList.set(i, imgList.get(i).substring(0,end));
                              }
                    }

                    return imgList;
            }

      private void endPreviewer(){
            this.dispose();
      }
}
```

```
import java.util.Arrays;


public class FormatHelper
{
	String[] selfClosing = {"meta", "link", "input"};
	String[] normalTags = {"b", "i", "a", "header", "img", "table", "ol", "dd", "dt", "dl", "li", "td",
"tr"};


	public FormatHelper()
	{
	}

	//assume in is well formed
	public String formatTabbedString(String in)
	{
		int numTabs = 0;
		boolean inTag = false;
		//find instance of tag, open and end.  pass that string into another function to
format
		String result = "";

		int start = in.indexOf('<');
		int end = in.indexOf('>');

		if(start == -1 || end == -1){
			System.out.println("No well formed tags found");
			return in;
		}

		//Add text before first tag
		if( start > 0){
			result += in.substring(0, start);
		}

		//start loop
		while( start > -1 && end > -1 ){
			String tag = in.substring(start + 1,  end); // the string between the two
tags found

			boolean isOpen = true;
```

```
if(tag.indexOf('/') == 0){
        //This tag is a closing tag
        tag = tag.substring(1);
        isOpen = false;
        if(inTag){
                inTag = false;
        }else{
                if(numTabs != 0)
                        numTabs--;
        }
}else{
        if(inTag){
                numTabs++;
        }else{
                inTag = true;
        }
}

if(numTabs < 0){
        System.out.println("Too many close tags");
        return in;
}

//IS IT A VALID TAG
if(tag.indexOf('=') > -1){ // tag is either A or IM
        if(tag.indexOf("a href=") == 0){
                //tag is a link
        }else if(tag.indexOf("img src=") == 0){
                //tag is an img
        }else{
                System.out.println("Found non-supported tag: " + tag);
                return in;
        }
}else if(Arrays.asList(normalTags).contains(tag)){
        //Tag is a normal tag
}else if(Arrays.asList(selfClosing).contains(tag)){
        //Tag is self closing
        numTabs--;
        if(numTabs == -1)
                numTabs = 0;
}else{
        System.out.println("Found non-supported tag: " + tag);
        return in;
```

```
				}

				//Tag is valid, oepnTag ( true = open, false = close )

				result += formatTag(tag, numTabs, isOpen);

				//Update start values
				start = in.indexOf('<', end + 1);
				end = in.indexOf('>', end + 1);

				if((end == -1 && start > -1) || (start == -1 && end > -1)){
					System.out.println("Missmatch tags");
					return in;
				}
			}
			//end loop

			//ADD REMIAINING STRING TO END OF RESULT
			int p = -1;
			end = in.indexOf('>');
			while(end > -1){
				p = end;
				end = in.indexOf('>', end + 1);
			}

			if(p + 1 < in.length()){
			result += in.substring(p);
			}

			return result;
		}

		private String formatTag(String tag, int numTabs, Boolean isOpen){
			String result = "";
			if(isOpen){
				result += getTabs(numTabs) + '<' + tag + '>' + '\n';
			}else{
				result += getTabs(numTabs) + "</" + tag + '>' + '\n';
			}
			return result;
		}
```

```
	private String getTabs(int i){
		String result = "";
		for(int k = 0; k < i; k++){
			result += '\t';
		}
		return result;
	}

}

import java.awt.BorderLayout;
import java.awt.Dimension;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.WindowConstants;
import javax.swing.border.EmptyBorder;

@SuppressWarnings("serial")
public class MainView extends JFrame
{
	InputHandler input;

	MenuView menu;
	ViewListener listener;
	BtnView buttons;
	JPanel panel;
	TabView tabView;
	LinkedView linkedView;

	public MainView(InputHandler i)
	{
		input = i;

		this.setMinimumSize(new Dimension(650,450));
		this.setTitle("Editor");
		this.setDefaultCloseOperation(EXIT_ON_CLOSE);

		listener = new ViewListener(input);

		menu = new MenuView(this, listener); // menuBar object
		buttons = new BtnView(this, listener);		// all of the buttons
```

```
                // make a new panel, give it a border with padding, select the Border layout
                panel = new JPanel();
                panel.setBorder(new EmptyBorder(0,15,15,15));
                panel.setLayout(new BorderLayout(0,0));
                this.setContentPane(panel);

                tabView = new TabView(this, listener);
                tabView.setSize(new Dimension(800,400));

                panel.add(tabView.getTabPane());

        // add each item to the content panel
        this.setJMenuBar(menu);
        panel.add(buttons, BorderLayout.NORTH);

        //this.setMinimumSize(new Dimension(300,300));
                this.pack();
                this.setLocationRelativeTo(null);
                this.setVisible(true);


    this.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
                this.addWindowListener(new java.awt.event.WindowAdapter() {
                        public void windowClosing(java.awt.event.WindowEvent windowEvent) {
                    listener.input.quit();
                        }
                });
        }

        public void quit(){
                if( tabView.closeAll() ){
                        //close program
                        this.dispose();
                        System.exit(0);
                }
        }

        public InputHandler getInputHandler()
        {
                return input;
        }

        public void addTab(String name, int id){
```

```
        tabView.createNewTab(name, id);
}

public String getText(){
        return tabView.getText();
}

public void setText(String text){
        tabView.setText(text);
}

public int getCursorStart(){
        return tabView.getCursorStart();
}

public int getCursorEnd(){
        return tabView.getCursorEnd();
}

public void setIsSaved(boolean b){
        input.setIsSaved(b);
}

public void newLinkedView(int strategy ){
        if(strategy == 1){
                linkedView = new LinkedView(this, new SortByAppear(), getText());
        }
        else{
                linkedView = new LinkedView(this, new SortByAlpha(), getText());
        }
}

public void updateFileName(String name){
        tabView.updateFileName(name);
}

public void toggleWordWrap(){
        tabView.toggleWordWrap();
}

public void setCursorStart(int n){
        tabView.setCursorStart(n);
}
```

```
}

import java.awt.event.ActionEvent;

import javax.swing.AbstractAction;
import javax.swing.Action;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.KeyStroke;

/**
 * Methods to display and run dropdown menu functions (save, load, button commands)
 * @author Dylan
 */
@SuppressWarnings("serial")
public class MenuView extends JMenuBar
{
	JMenu file;
	JMenuItem newDoc;
	JMenuItem open;
	JMenuItem save;
	JMenuItem saveAs;
	JMenuItem exit;

	JMenu edit;
	JMenuItem undo;
	JMenuItem redo;

	JMenu view;
	JMenuItem chooseSort;
	JMenuItem imgPreview;
	JMenuItem tagFormatter;

	/**
	 * The constructor for the MenuView class.
	 * @param parent
	 * @param listener
	 */
	public MenuView(final MainView parent, ViewListener listener){
		this.setSize(parent.getWidth(), 25);

		Action actionNew = new AbstractAction("new") {
```

```
                        public void actionPerformed(ActionEvent e){}
                };
                Action actionOpen = new AbstractAction("Open File...") {
                        public void actionPerformed(ActionEvent e){}
                };
                Action actionSave = new AbstractAction("Save") {
                        public void actionPerformed(ActionEvent e){}
                };
                Action actionSaveAs = new AbstractAction("Save As...") {
                        public void actionPerformed(ActionEvent e){}
                };
                Action actionExit = new AbstractAction("Exit") {
                        public void actionPerformed(ActionEvent e){}
                };

                actionNew.putValue(Action.ACCELERATOR_KEY,
KeyStroke.getKeyStroke("control N"));
                actionOpen.putValue(Action.ACCELERATOR_KEY,
KeyStroke.getKeyStroke("control O"));
                actionSave.putValue(Action.ACCELERATOR_KEY,
KeyStroke.getKeyStroke("control S"));
                actionSaveAs.putValue(Action.ACCELERATOR_KEY,
KeyStroke.getKeyStroke("control E"));
                actionExit.putValue(Action.ACCELERATOR_KEY,
KeyStroke.getKeyStroke("control Q"));

                file = new JMenu();

                newDoc = new JMenuItem(actionNew);
                newDoc.setText("New");
                newDoc.addActionListener(listener);

                open = new JMenuItem(actionOpen);
                open.setText("Open File...");
                open.addActionListener(listener);

                save = new JMenuItem(actionSave);
                save.setText("Save");
                save.addActionListener(listener);

                saveAs = new JMenuItem(actionSaveAs);
                saveAs.setText("Save As...");
                saveAs.addActionListener(listener);
```

```
		exit = new JMenuItem(actionExit);
		exit.setText("Exit");
		exit.addActionListener(listener);

		file.setText("File");

		file.add(newDoc);
		file.add(open);
		file.add(save);
		file.add(saveAs);
		file.add(exit);

		this.add(file);


		Action actionUndo = new AbstractAction("Undo") {
			public void actionPerformed(ActionEvent e){}
		};
		Action actionRedo = new AbstractAction("Redo") {
			public void actionPerformed(ActionEvent e){}
		};

		actionUndo.putValue(Action.ACCELERATOR_KEY,
KeyStroke.getKeyStroke("control Z"));
		actionRedo.putValue(Action.ACCELERATOR_KEY,
KeyStroke.getKeyStroke("control Y"));

		edit = new JMenu();

		undo = new JMenuItem(actionUndo);
		undo.setText("Undo");
		undo.addActionListener(listener);

		redo = new JMenuItem(actionRedo);
		redo.setText("Redo");
		redo.addActionListener(listener);

		edit.setText("Edit");
		edit.add(undo);
		edit.add(redo);
		this.add(edit);


		Action actionChooseSort = new AbstractAction("Linked view") {
			public void actionPerformed(ActionEvent e){}
		};
		Action actionImgPreview = new AbstractAction("Preview image") {
			public void actionPerformed(ActionEvent e){}
		};
		Action actionTagFormatter = new AbstractAction("Tag layout") {
			public void actionPerformed(ActionEvent e){}
		};

		actionChooseSort.putValue(Action.ACCELERATOR_KEY,
KeyStroke.getKeyStroke("control L"));
		actionImgPreview.putValue(Action.ACCELERATOR_KEY,
KeyStroke.getKeyStroke("control I"));
		actionTagFormatter.putValue(Action.ACCELERATOR_KEY,
KeyStroke.getKeyStroke("control T"));

		view = new JMenu();

		chooseSort = new JMenuItem(actionChooseSort);
		chooseSort.setText("Linked view");
		chooseSort.addActionListener(listener);

		imgPreview = new JMenuItem(actionImgPreview);
		imgPreview.setText("Preview image");
		imgPreview.addActionListener(listener);

		tagFormatter = new JMenuItem(actionTagFormatter);
		tagFormatter.setText("Tag layout");
		tagFormatter.addActionListener(listener);

		view.setText("View");
		view.add(chooseSort);
		view.add(imgPreview);
		view.add(tagFormatter);

		this.add(view);


		this.setVisible(true);
```

```
        }
}

import java.awt.FlowLayout;

import javax.swing.JButton;
import javax.swing.JPanel;

/**
 * Methods to display buttons and run attached functions (inserts, HTML constructs)
 * @author Dylan, Andrew
 *
 */
public class BtnView extends JPanel
{
        JButton btnA;//<a>
        JButton btnB;//<b>   (bold)
        JButton btnI;//<i>   (italics)
        JButton btnHeader;//<header>
        JButton btnOl;//<ol>   (ordered list)
        JButton btnUl;//<ul>   (unordered list)
        JButton btnDl;//<dl>   (dictionary list)
        JButton btnTable;//<Table>
        JButton btnImg;//<img>   (Image)

        ViewListener vListener;

        /**
         * Creates and connects all of the buttons to the listeners.
         * @param parent
         * @param listener
         */
        public BtnView(MainView parent, ViewListener listener){

                vListener = listener;

                this.setSize(parent.getWidth(), 25);
                this.setLayout(new FlowLayout());

                this.add(btnB = new JButton("b"));
                btnB.setFocusable(false);
                btnB.addActionListener(vListener);
```

```
		this.add(btnI = new JButton("i"));
		btnI.addActionListener(vListener);
		btnI.setFocusable(false);

		this.add(btnA = new JButton("a"));
		btnA.addActionListener(vListener);
		btnA.setFocusable(false);

		this.add(btnHeader = new JButton("header"));
		btnHeader.addActionListener(vListener);
		btnHeader.setFocusable(false);

		this.add(btnOl = new JButton("ol"));
		btnOl.addActionListener(vListener);
		btnOl.setFocusable(false);

		this.add(btnUl = new JButton("ul"));
		btnUl.addActionListener(vListener);
		btnUl.setFocusable(false);

		this.add(btnDl = new JButton("dl"));
		btnDl.addActionListener(vListener);
		btnDl.setFocusable(false);

		this.add(btnTable = new JButton("table"));
		btnTable.addActionListener(vListener);
		btnTable.setFocusable(false);

		this.add(btnImg = new JButton("img"));
		btnImg.addActionListener(vListener);
		btnImg.setFocusable(false);

		this.setVisible(true);
	}
}

import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextArea;
import javax.swing.JViewport;
import javax.swing.ScrollPaneConstants;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

@SuppressWarnings("serial")
public class TabView extends JPanel{

        MainView mainView;
        ViewListener listener;
        JTabbedPane tabPane;
        static Image img;

        public TabView(MainView parent, ViewListener vListener){
                mainView = parent;
                listener = vListener;
                tabPane = new JTabbedPane();

                img = null;
                //get icon
                try {
                   img = ImageIO.read(getClass().getResource("resources/closeIcon.png"));
                } catch (IOException ex) {}

                tabPane.addChangeListener(new ChangeListener() {
                        public void stateChanged(ChangeEvent e) {
                                if( tabPane.getTabCount() < 1 ){return;}
                                mainView.input.changeCurrentFile(tabPane.getSelectedIndex());
                        }
                });
        }
```

```
public JTabbedPane getTabPane(){
	return tabPane;
}

public String getText(){
	JTextArea textArea = getTextArea();

	if(textArea == null){
		return ""; // no tabs exist
	}

	return textArea.getText();
}

public void setText(String text){
	JTextArea textArea = getTextArea();

	if(textArea == null){
		return; // no tabs exist
	}

	textArea.setText(text);
}

private JTextArea getTextArea(){
	int index = tabPane.getSelectedIndex();

	if(index == -1){
		return null; // no tabs exist
	}

	JPanel tab = (JPanel)tabPane.getComponentAt(index);
	JScrollPane scrollPane = (JScrollPane)tab.getComponent(0);
	JViewport viewport = (JViewport)scrollPane.getComponent(0);
	JTextArea textArea = (JTextArea)viewport.getComponent(0);

	return textArea;
}

public int getCursorStart(){
	JTextArea textArea = getTextArea();

	if(textArea == null){
			return -1; // no tabs exist
		}

		return textArea.getCaret().getDot();
	}

	public void setCursorStart(int n){
		JTextArea textArea = getTextArea();

		if(textArea == null){
			return; // no tabs exist
		}

		textArea.getCaret().setDot(n);
	}

	public int getCursorEnd(){
		JTextArea textArea = getTextArea();

		if(textArea == null){
			return -1; // no tabs exist
		}
		return textArea.getCaret().getMark();
	}

	public void updateFileName(String name){
		int index = tabPane.getSelectedIndex();
		JPanel innerPane = (JPanel)tabPane.getTabComponentAt(index);
		JLabel label = (JLabel)innerPane.getComponent(0);
		label.setText(name);
	}

	public void toggleWordWrap(){
		if(getTextArea().getWrapStyleWord()){
			getTextArea().setWrapStyleWord(false);
		}

		else getTextArea().setWrapStyleWord(true);
	}

	public void createNewTab(String name, int index){
		JPanel innerPane = new JPanel();
```

```
                TextAreaView textView = new TextAreaView(mainView, listener);
                textView.getTextArea().addKeyListener( new KeyListener(){
                         public void keyReleased(KeyEvent e) {
                         }

                        @Override
                        public void keyTyped(KeyEvent e) {
                                if(e.getKeyChar() == '\n'){
                                        mainView.getInputHandler().buttonViewInput("update");
                                }

                                if( e.getKeyChar() == '\b'){

mainView.getInputHandler().buttonViewInput("Subtractive");
                                }
                        }

                        @Override
                        public void keyPressed(KeyEvent e) {
                        }
                });

                JScrollPane scrollPane = new JScrollPane(textView.getTextArea());
                scrollPane.setName("scrollPane");

scrollPane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_N
EEDED);

scrollPane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_
NEVER);

            innerPane.setName(Integer.toString(index));
                innerPane.add(scrollPane);
                innerPane.setLayout(new GridLayout(1,2));

                tabPane.addTab(name, innerPane);
                tabPane.setSelectedIndex(tabPane.getTabCount()-1);

                //set title with close button
                tabPane.setTabComponentAt(
                                tabPane.getTabCount()-1,
                                getTitlePanel(tabPane, innerPane, name, index,
mainView.getInputHandler())
```

```
		);
	}

	// closes all tabs return true if all are closed, false if user aborted the quit
	public boolean closeAll(){
		int tabCount = tabPane.getTabCount();
		for(int i = 0; i < tabCount; i++){
			JPanel tab = (JPanel)tabPane.getComponentAt(i);
			int id = Integer.parseInt(tab.getName());
			if(mainView.getInputHandler().closeTab(id)){
				tabPane.remove(tab);
				if(tabPane.getTabCount() == 0){
					return true;
				}
				i = -1;
			}else{
				return false;
			}
		}
		return true;
	}

	private static JPanel getTitlePanel(final JTabbedPane tabbedPane, final JPanel panel,
String title, final int id, final InputHandler input){
		JPanel titlePanel = new JPanel();
		titlePanel.setOpaque(false);
		JLabel titleLbl = new JLabel(title);
		titleLbl.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 5));
		titlePanel.add(titleLbl);
		JButton closeButton = new JButton();
		closeButton.setOpaque(true);
		int size = 14;

		if(img != null){
			closeButton.setIcon(new ImageIcon(img));
		}else{
			closeButton.setText("x");
		}

		closeButton.setPreferredSize(new Dimension(size, size));

		closeButton.addActionListener( new ActionListener(){
			@Override
```

```
                        public void actionPerformed(ActionEvent e) {
                            if(input.closeTab(id))
                                tabbedPane.remove(panel);
                        }
                    });

            titlePanel.add(closeButton);

            return titlePanel;
        }
}

import javax.swing.JTextArea;

/**
 * Class that deals with the text area front end.
 * @author Andrew
 *
 */
@SuppressWarnings("serial")
public class TextAreaView extends JTextArea{

    ViewListener vListener;
    MainView mainView;

    JTextArea textArea;
    String lastCharIn;
    int prevCharPos;

    /**
     * The constructor of the TestAreaView.
     * @param parent
     * @param listener
     */
    public TextAreaView(MainView parent, ViewListener listener){
        vListener = listener;
        mainView = parent;

        textArea = new JTextArea(50, 50);
        textArea.setWrapStyleWord(true);
  textArea.setLineWrap(true);
    }
	/**
	 * Returns the text area
	 * @return
	 */
	public JTextArea getTextArea(){
		return textArea;
	}

	/**
	 * Returns the cursor start position.
	 * @return
	 */
	public int getCursorStart(){
		return textArea.getCaret().getDot();
	}

	/**
	 * Returns the cursor end position.
	 * @return
	 */
	public int getCursorEnd(){
		return textArea.getCaret().getMark();
	}
}

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JMenuItem;

/**
 * Methods to get and relay updates to the other view files
 * @author Dylan
 *
 */
public class ViewListener implements ActionListener
{
	InputHandler input;

	/**
	 * The constructor of the ViewListener class.
	 * @param i
	 */
	public ViewListener(InputHandler i){
		input = i;
	}

	/**
	 * Listeners for the Buttons and Menu items.
	 */
	public void actionPerformed(ActionEvent arg0)
	{
		//ButtonView
		if(arg0.getSource().getClass().isAssignableFrom((new JButton()).getClass())){
			String txt = ((JButton) arg0.getSource()).getText();
			input.buttonViewInput(txt);
		}

		//MenuView
		if(arg0.getSource().getClass().isAssignableFrom((new 
JMenuItem()).getClass())){
			String txt = ((JMenuItem) arg0.getSource()).getText();
			input.menuViewInput(txt);
		}
	}
}
```