Criterion C: Development

Development Process Techniques

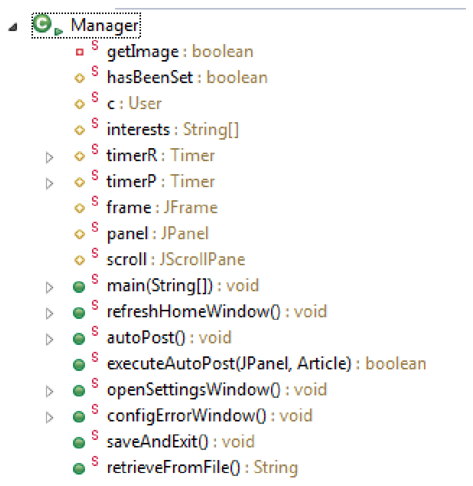
Developed in the Eclipse IDE, my Java project utilizes many different techniques including:

- User-defined classes
    - Manager
    - User
    - Article, Source, and Category
    - Word
- User-defined methods and algorithms
    - Refreshing news
    - JSON to Java conversion
    - Article keyword generation and filtering
    - Writing to and reading from file
    - Input of name and settings
    - Settings window generation
    - Generation of error windows (message length, configuration)
    - Credential validation
    - Auto-posting
    - Posting to Facebook
    - Posting to Twitter
- Data Structures
    - ArrayList
    - TreeSet and TreeMap
- Libraries
    - java.swing and java.awt
    - java.io
    - java.net
    - Gson
    - restfb
    - twitter4j
- API's
    - News API

Classes and Methods

This project contains 6 classes: Manager (which is executed and contains the main method), User (contains all user settings, news, and posting methods), Article, Source, Category, and Word.

Manager
This class is executed and contains the main method. The fields getImage, hasBeenSet, User c, interests, both timers, frame, panel, and scroll are accessed throughout the Manager class. At the beginning of development I would pass the variables (set in main) to the other methods, but this became inefficient.

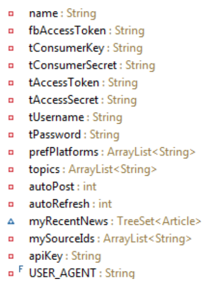


```
Manager
  getImage : boolean
  hasBeenSet : boolean
  c : User
  interests : String[]
  timerR : Timer
  timerP : Timer
  frame : JFrame
  panel : JPanel
  scroll : JScrollPane
  main(String[]) : void
  refreshHomeWindow() : void
  autoPost() : void
  executeAutoPost(JPanel, Article) : boolean
  openSettingsWindow() : void
  configErrorWindow() : void
  saveAndExit() : void
  retrieveFromFile() : String
```

User

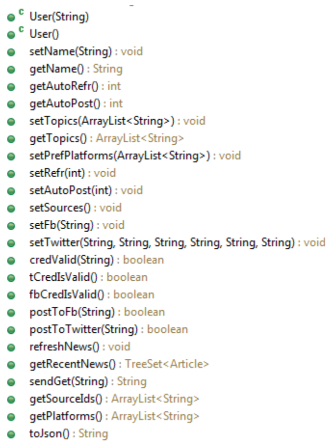
The User class contains all personal preferences, the keys and access tokens for social media platforms and connection to the internet, and methods for modifying and accessing these fields and posting to Facebook and Twitter.

Fields



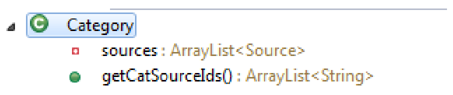
```
name : String
fbAccessToken : String
tConsumerKey : String
tConsumerSecret : String
tAccessToken : String
tAccessSecret : String
tUsername : String
tPassword : String
prefPlatforms : ArrayList<String>
topics : ArrayList<String>
autoPost : int
autoRefresh : int
myRecentNews : TreeSet<Article>
mySourceIds : ArrayList<String>
apiKey : String
USER_AGENT : String
```

Methods

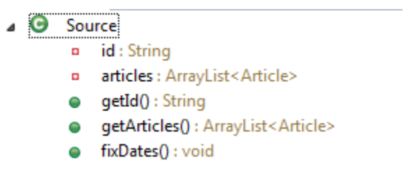


```
User(String)
User()
setName(String) : void
getName() : String
getAutoRefr() : int
getAutoPost() : int
setTopics(ArrayList<String>) : void
getTopics() : ArrayList<String>
setPrefPlatforms(ArrayList<String>) : void
setRefr(int) : void
setAutoPost(int) : void
setSources() : void
setFb(String) : void
setTwitter(String, String, String, String, String, String) : void
credValid(String) : boolean
tCredIsValid() : boolean
fbCredIsValid() : boolean
postToFb(String) : boolean
postToTwitter(String) : boolean
refreshNews() : void
getRecentNews() : TreeSet<Article>
sendGet(String) : String
getSourceIds() : ArrayList<String>
getPlatforms() : ArrayList<String>
toJson() : String
```

Category, Source, and Article



```
Category
  sources : ArrayList<Source>
  getCatSourceIds() : ArrayList<String>
```



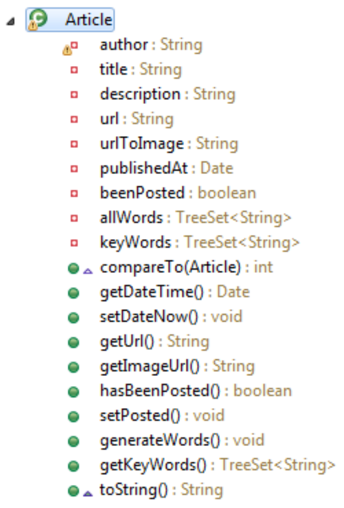
```
Source
  id : String
  articles : ArrayList<Article>
  getId() : String
  getArticles() : ArrayList<Article>
  fixDates() : void
```

The Category and Source classes were written to hold information while in the process of getting Article information. The initial call to NewsAPI gets the sources in a specific category – a Category object is made from that response. It contains an ArrayList of incomplete Source objects (they are not filled with Articles yet), but is needed to get the source ID’s (such as “bbc-sport” or “buzzfeed”). For each source ID, the User class makes a Source for it containing an ArrayList of Articles. The original Category objects were only used to access the source ID’s, and do not contain the new complete Source objects.

All of these classes do not have a constructor written out because they are created by deserializing JSON strings (in the main method of

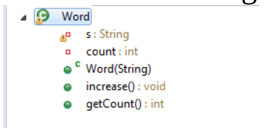
Manager).

The Article class contains information for that article, set and get methods, and methods that generate the Article's keywords. It also implements the Comparable<T> interface and overrides compareTo() and toString().



```
Article
    author : String
    title : String
    description : String
    url : String
    urlToImage : String
    publishedAt : Date
    beenPosted : boolean
    allWords : TreeSet<String>
    keyWords : TreeSet<String>
    compareTo(Article) : int
    getDateTime() : Date
    setDateNow() : void
    getUrl() : String
    getImageUrl() : String
    hasBeenPosted() : boolean
    setPosted() : void
    generateWords() : void
    getKeyWords() : TreeSet<String>
    toString() : String
```

Word
This simple class holds a string and the integer count of how many times it has been repeated. This class is used in determining the most common keywords from a set of articles.

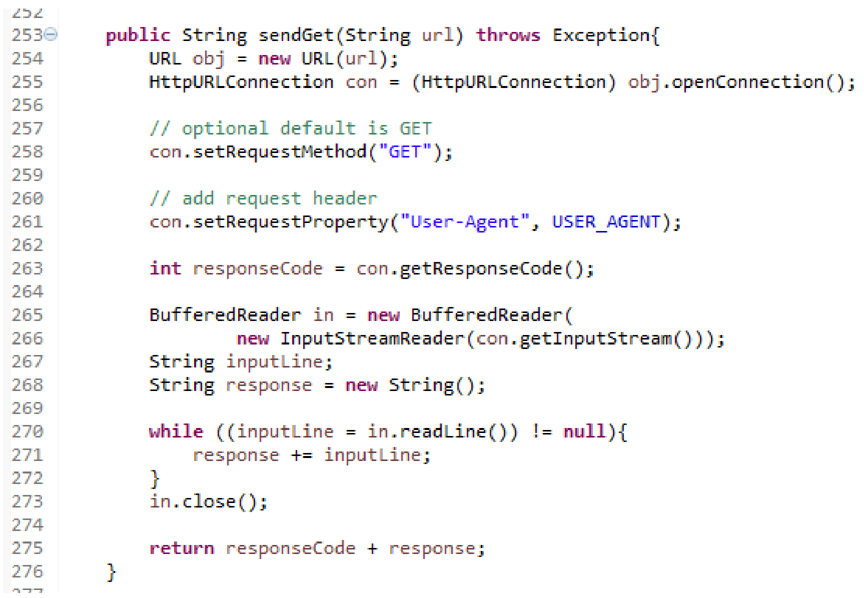


```
Word
    s : String
    count : int
    Word(String)
    increase() : void
    getCount() : int
```

Data Structures, Algorithmic Thinking, and Libraries

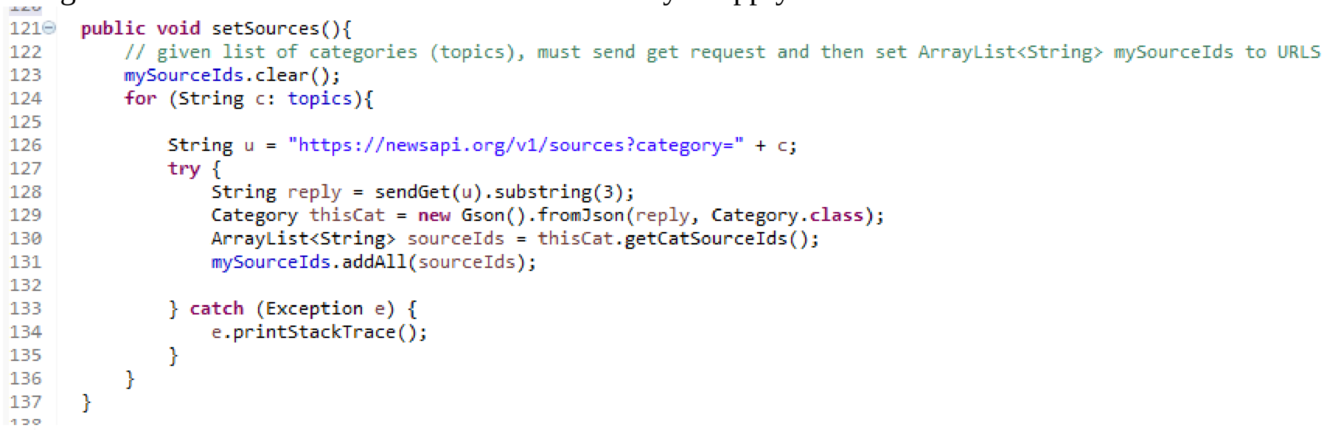
The base of this product is its ability to retrieve news sources using News API (https://newsapi.org/). The list of interest categories (ArrayList<String> topics in User) must then be converted to Category objects based on the NewsAPI response, from which source IDs are pulled to send yet another GET response to get the top or latest articles from that source.

This is the sendGet(String url) method in the User class that uses a URL object and HttpURLConnectiono object to send a request to a String url and read it in with an InputStreamReader and BufferedReader object. It is used by both the setSources() and refreshNews() methods.

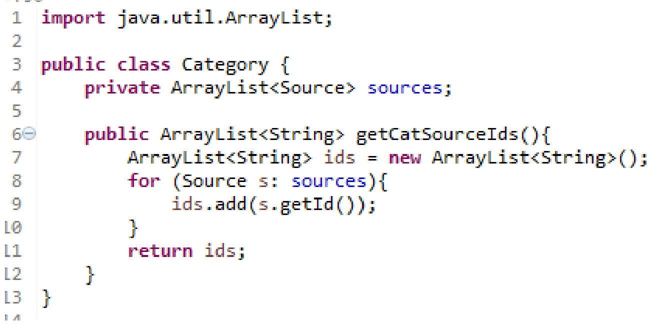


```java
    public String sendGet(String url) throws Exception{
        URL obj = new URL(url);
        HttpURLConnection con = (HttpURLConnection) obj.openConnection();

        // optional default is GET
        con.setRequestMethod("GET");

        // add request header
        con.setRequestProperty("User-Agent", USER_AGENT);

        int responseCode = con.getResponseCode();

        BufferedReader in = new BufferedReader(
                new InputStreamReader(con.getInputStream()));
        String inputLine;
        String response = new String();

        while ((inputLine = in.readLine()) != null){
            response += inputLine;
        }
        in.close();

        return responseCode + response;
    }
```

This is the setSources() method, which iterates through the ArrayList<String> topics, calling the sendGet(String url) to the category's information. The Gson library is used to convert this String response (in a JSON format) to a Category object, also shown below. The Gson library is a Java serialization/deserialization library that was the best choice for this project because it could be used to both deserialize the JSON response from NewsAPI into Java objects and serialize the User object into a JSON string to write to a file on exit. It was incredible easy to apply in both cases.

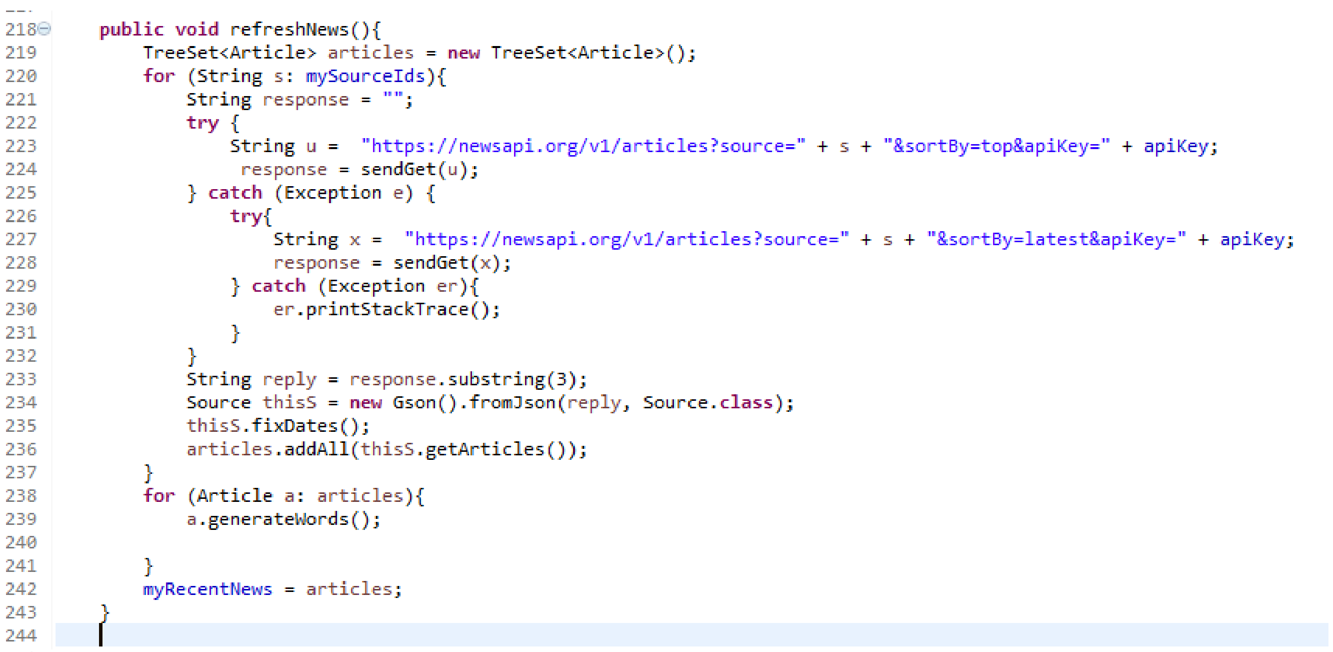


```java
public void setSources(){
    // given list of categories (topics), must send get request and then set ArrayList<String> mySourceIds to URLS
    mySourceIds.clear();
    for (String c: topics){

        String u = "https://newsapi.org/v1/sources?category=" + c;
        try {
            String reply = sendGet(u).substring(3);
            Category thisCat = new Gson().fromJson(reply, Category.class);
            ArrayList<String> sourceIds = thisCat.getCatSourceIds();
            mySourceIds.addAll(sourceIds);

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

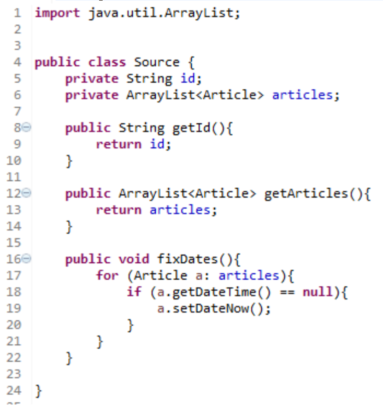


```java
import java.util.ArrayList;

public class Category {
    private ArrayList<Source> sources;

    public ArrayList<String> getCatSourceIds(){
        ArrayList<String> ids = new ArrayList<String>();
        for (Source s: sources){
            ids.add(s.getId());
        }
        return ids;
    }
}
```

The larger refreshNews() method retrieves articles based on mySourceIds, which were set by the setSources() method. For every source ID, a call to the sendGet(String url) is used to retrieve either the top articles for that source, or if that doesn't work, the latest articles. A try/catch block is used to specify a response if an exception is thrown. Next, the response code (3 characters), that is not useful in creating the Source objects and therefore Article objects, is cut off the response and the resulting string is saved and deserialized into a Source object, which contains an ArrayList of Article objects that are automatically created as well. The article's dates are fixed if null, and set to the current date with the setDateNow() method in the Article class. All the articles from this particular Source are added into the TreeSet<Article> articles, and at the end of this method, myRecentNews is set to articles. This is done so that in case the news retrieval encounters errors and stops, myRecentNews would not be affected. Because I did not want duplicates of an article, I chose to use a Set for both articles and myRecentNews, and a TreeSet because I wanted the Articles to be able to be ordered by date. At first I had planned for there to be an allNews collection that would accumulate all the news in a set period (possibly a few days), but as I got further into development I realized that there was little use for it and could actually cause memory problems if allowed to grow too large. Thus, I took it out.

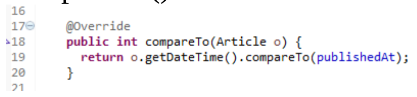


```java
    public void refreshNews(){
        TreeSet<Article> articles = new TreeSet<Article>();
        for (String s: mySourceIds){
            String response = "";
            try {
                String u =  "https://newsapi.org/v1/articles?source=" + s + "&sortBy=top&apiKey=" + apiKey;
                 response = sendGet(u);
            } catch (Exception e) {
                try{
                    String x =  "https://newsapi.org/v1/articles?source=" + s + "&sortBy=latest&apiKey=" + apiKey;
                    response = sendGet(x);
                } catch (Exception er){
                    er.printStackTrace();
                }
            }
            String reply = response.substring(3);
            Source thisS = new Gson().fromJson(reply, Source.class);
            thisS.fixDates();
            articles.addAll(thisS.getArticles());
        }
        for (Article a: articles){
            a.generateWords();

        }
        myRecentNews = articles;
    }

```



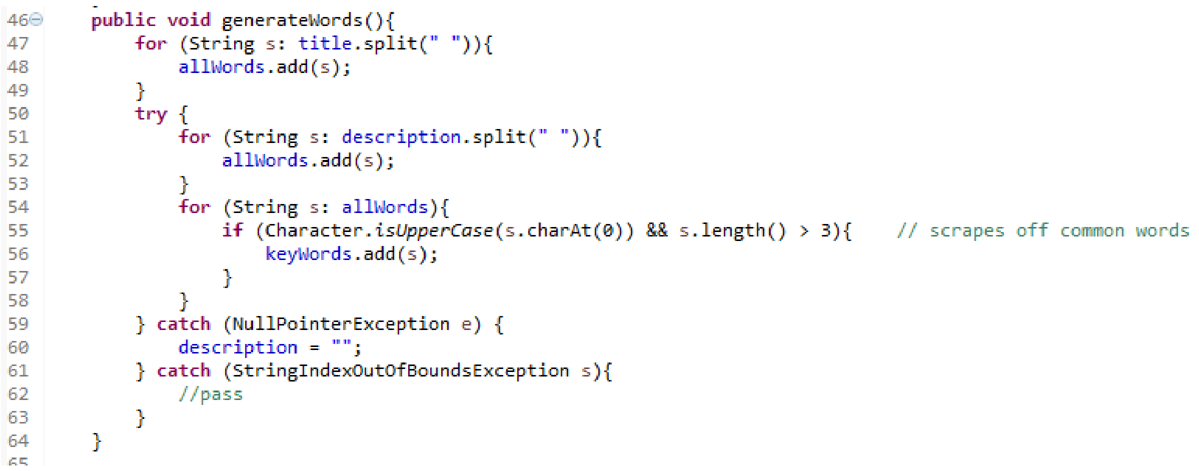
```java
import java.util.ArrayList;


public class Source {
    private String id;
    private ArrayList<Article> articles;

    public String getId(){
        return id;
    }

    public ArrayList<Article> getArticles(){
        return articles;
    }

    public void fixDates(){
        for (Article a: articles){
            if (a.getDateTime() == null){
                a.setDateNow();
            }
        }
    }

}
```

The Article class implements the Comparable<T> interface so that the TreeSet<Article> orders them based on date. The superclass compareTo() method is overridden with one that compares the Article to another based on their publishedAt Dates. The Article class also has setter and getter methods that are used in its compareTo() method and for many other purposes.



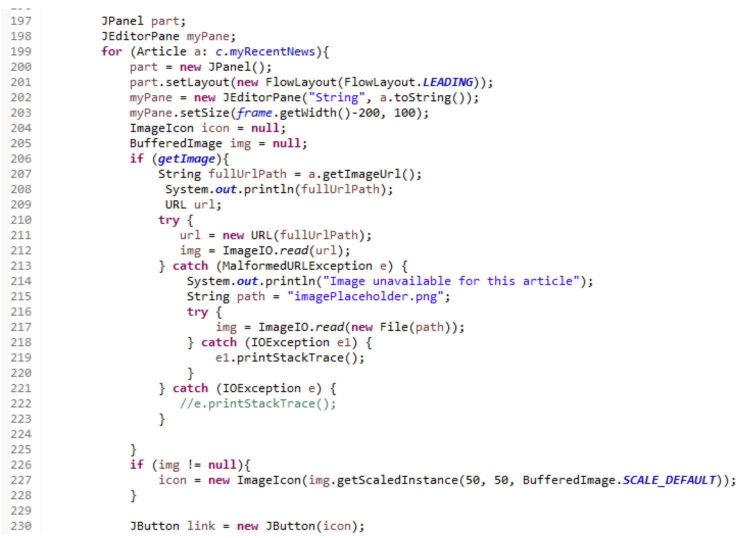
```java

    @Override
    public int compareTo(Article o) {
      return o.getDateTime().compareTo(publishedAt);
    }
```

After the Articles are added to the TreeSet<Article> articles, articles is iterated over and the generateWords() method is called on each of the Article objects. These keywords will be used in the auto-posting process. A NullPointerException catch is needed as some articles come without a description, and a StringIndexOutOfBoundsException catch is also needed to deal with oddly-written responses from NewsAPI. Once the title and description have been split into their component words, capitalized words that are over 4 or more letters long are added to the keyWords TreeSet (being a Set prevents duplicates). This filtering prevents common words such as “the” or “A” from being considered when determining the most common words in myRecentNews, as is done in the auto-post process.



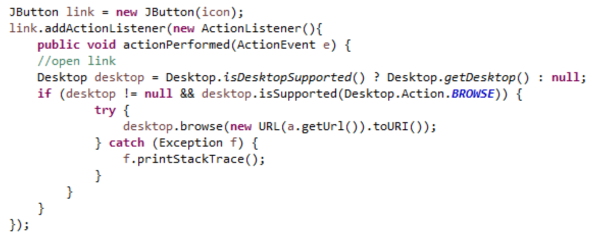
```java
public void generateWords(){
    for (String s: title.split(" ")){
        allWords.add(s);
    }
    try {
        for (String s: description.split(" ")){
            allWords.add(s);
        }
        for (String s: allWords){
            if (Character.isUpperCase(s.charAt(0)) && s.length() > 3){    // scrapes off common words
                keyWords.add(s);
            }
        }
    } catch (NullPointerException e) {
        description = "";
    } catch (StringIndexOutOfBoundsException s){
        //pass
    }
}
```

These articles are displayed by a call in the main method of the Manager class to refreshHomeWindow(), which first executes refreshNews() on the User c. For each of the articles in c.myRecentNews, a JPanel is developed which contains a button (with the article's image or a placeholder image) to open the link, a JEditorPane with the article's information, and a "Post about it!" button (the poster JButton). This code sets up the JEditorPane myPane and retrieval of the image for the first button:



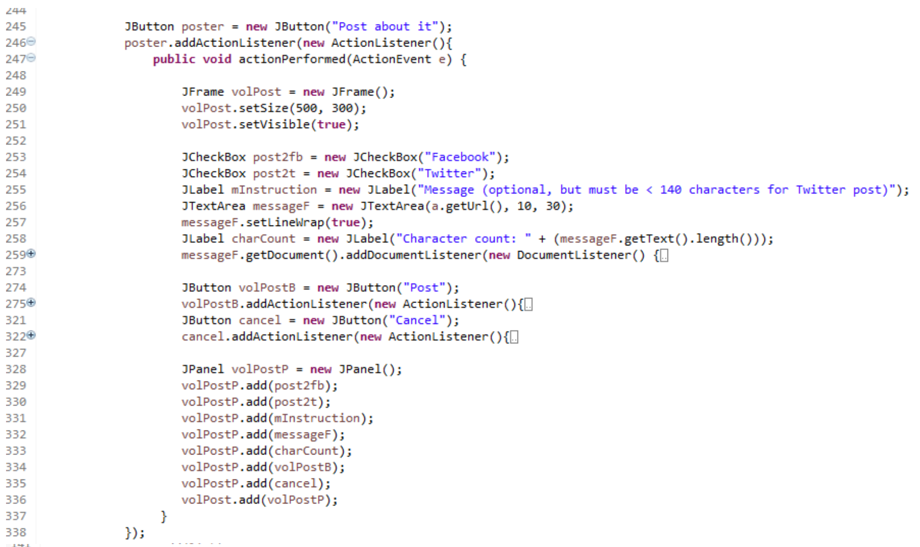
```java
JPanel part;
JEditorPane myPane;
for (Article a: c.myRecentNews){
    part = new JPanel();
    part.setLayout(new FlowLayout(FlowLayout.LEADING));
    myPane = new JEditorPane("String", a.toString());
    myPane.setSize(frame.getWidth()-200, 100);
    ImageIcon icon = null;
    BufferedImage img = null;
    if (getImage){
        String fullUrlPath = a.getImageUrl();
         System.out.println(fullUrlPath);
         URL url;
        try {
           url = new URL(fullUrlPath);
           img = ImageIO.read(url);
        } catch (MalformedURLException e) {
            System.out.println("Image unavailable for this article");
            String path = "imagePlaceholder.png";
            try {
                img = ImageIO.read(new File(path));
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        } catch (IOException e) {
           //e.printStackTrace();
        }

    }
    if (img != null){
        icon = new ImageIcon(img.getScaledInstance(50, 50, BufferedImage.SCALE_DEFAULT));
    }

    JButton link = new JButton(icon);
```

The link JButton is set (with an ActionListener) to open a browser window to the article itself, using the Desktop class. This class is useful because it allows a Java application to launch applications (in this case Google Chrome, Mozilla Firefox, Internet Explorer, or whatever your default browser is) to a specific URI (URL's are a subset of URI's). The conditional operator ? : is used to set desktop to Desktop.getDesktop() if the Desktop class if supported: if not, desktop is set to null. A try/catch block is used to either open the article or catch an Exception.

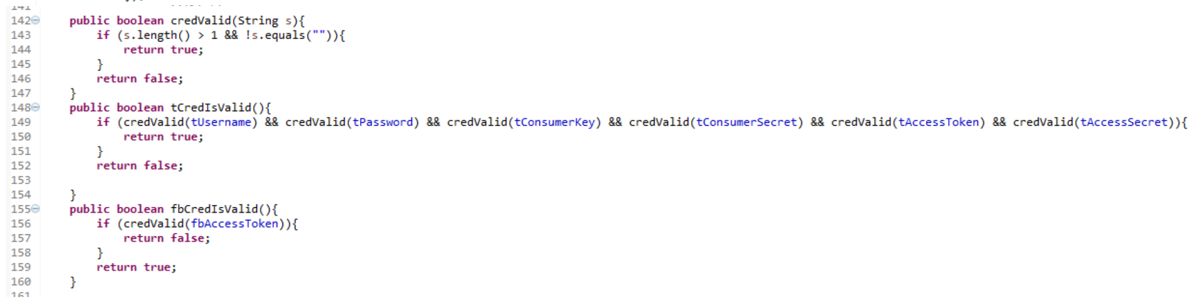


```java
JButton link = new JButton(icon);
link.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e) {
    //open link
    Desktop desktop = Desktop.isDesktopSupported() ? Desktop.getDesktop() : null;
    if (desktop != null && desktop.isSupported(Desktop.Action.BROWSE)) {
            try {
                desktop.browse(new URL(a.getUrl()).toURI());
            } catch (Exception f) {
                f.printStackTrace();
            }
        }
    }
});
```

The poster JButton uses an ActionListener to open a posting window containing JCheckbox's for Facebook and Twitter, a JLabel, and a JTextArea containing the article's URL that the user can type in. The JTextArea has a DocumentListener that, if anything is removed, inserted, or changed, makes the JLabel charCount set to the number of characters in the JTextArea. The ActionListener on the voluntary-post button (volPostB) calls the postToFb() and postToTwitter() methods of the User class, after checking for valid credentials and for the message's length (if Twitter is selected). In this case tests the credentials with fbCredIsValid() and/or tCredIsValid() for not being null or blank, as authorization is checked when trying to post and will throw an exception if the credentials are invalid. If the fields are blank, configErrorWindow() is called. These elements are all added to the voluntary-post panel, which is added to the window that opens if you click the "Post about it!" button.

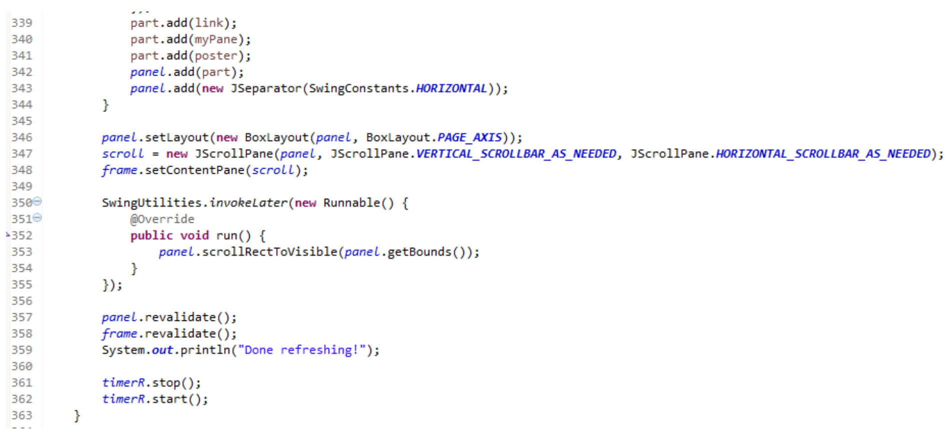


```java
        JButton poster = new JButton("Post about it");
        poster.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e) {

                JFrame volPost = new JFrame();
                volPost.setSize(500, 300);
                volPost.setVisible(true);

                JCheckBox post2fb = new JCheckBox("Facebook");
                JCheckBox post2t = new JCheckBox("Twitter");
                JLabel mInstruction = new JLabel("Message (optional, but must be < 140 characters for Twitter post)");
                JTextArea messageF = new JTextArea(a.getUrl(), 10, 30);
                messageF.setLineWrap(true);
                JLabel charCount = new JLabel("Character count: " + (messageF.getText().length()));
                messageF.getDocument().addDocumentListener(new DocumentListener() {□

                JButton volPostB = new JButton("Post");
                volPostB.addActionListener(new ActionListener(){□
                JButton cancel = new JButton("Cancel");
                cancel.addActionListener(new ActionListener(){□

                JPanel volPostP = new JPanel();
                volPostP.add(post2fb);
                volPostP.add(post2t);
                volPostP.add(mInstruction);
                volPostP.add(messageF);
                volPostP.add(charCount);
                volPostP.add(volPostB);
                volPostP.add(cancel);
                volPost.add(volPostP);
            }
        });
```



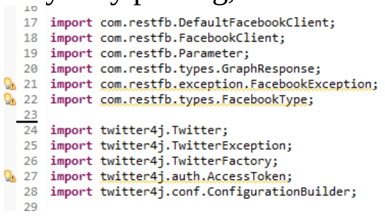
```java
    public boolean credValid(String s){
        if (s.length() > 1 && !s.equals("")){
            return true;
        }
        return false;
    }
    public boolean tCredIsValid(){
        if (credValid(tUsername) && credValid(tPassword) && credValid(tConsumerKey) && credValid(tConsumerSecret) && credValid(tAccessToken) && credValid(tAccessSecret)){
            return true;
        }
        return false;

    }
    public boolean fbCredIsValid(){
        if (credValid(fbAccessToken)){
            return false;
        }
        return true;
    }
```

After the poster JButton is created, components are added to the part JPanel, which is then added to the larger JPanel panel. JSeparators separate the articles in panel. The layout for panel is set to BoxLayout (in which components are stacked) along the page axis. A new JScrollPane scroll is created with the contents of panel in it, and frame is set to contain the scroll JScrollPane. The next few lines of code, that reset the scroll bar to the top, took a lot of trial and error to figure out. The most intuitive code that I tried at first was "scroll.getVerticalScrollBar().setValue(0);", but this does not work. Instead, the invokeLater(new Runnable(){…}) makes it so that the scroll bar will be set correctly (with .scrollRectToVisible()) only after all other pending events (that would interfere or make it scroll back to the bottom) have been processed. Panel and frame are revalidated, and the refresh timer is restarted.



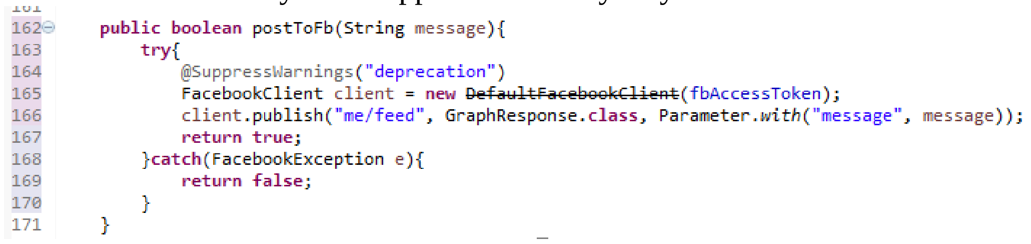
```java
            part.add(link);
            part.add(myPane);
            part.add(poster);
            panel.add(part);
            panel.add(new JSeparator(SwingConstants.HORIZONTAL));
        }

        panel.setLayout(new BoxLayout(panel, BoxLayout.PAGE_AXIS));
        scroll = new JScrollPane(panel, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        frame.setContentPane(scroll);

        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                panel.scrollRectToVisible(panel.getBounds());
            }
        });

        panel.revalidate();
        frame.revalidate();
        System.out.println("Done refreshing!");

        timerR.stop();
        timerR.start();
    }
```

The other main element of this application is the ability to post to Facebook and Twitter. In the case of Facebook, this is done by using RestFB. Twitter4j is used to post to Twitter. These two libraries allow for very easy posting, as shown by the brevity and simplicity of these methods.



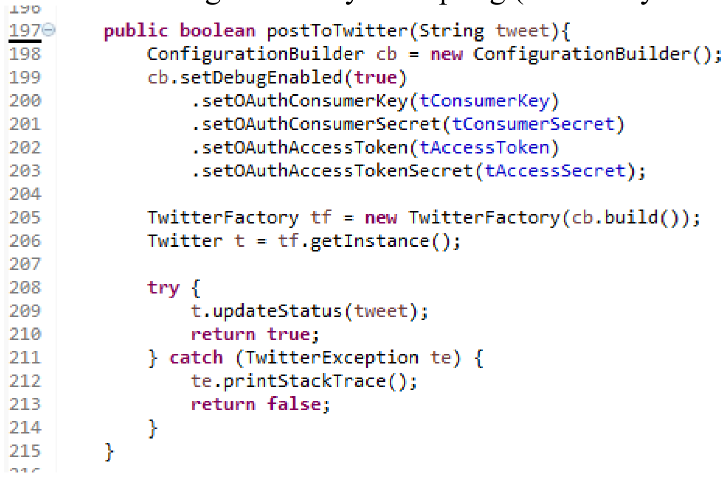
```java
import com.restfb.DefaultFacebookClient;
import com.restfb.FacebookClient;
import com.restfb.Parameter;
import com.restfb.types.GraphResponse;
import com.restfb.exception.FacebookException;
import com.restfb.types.FacebookType;

import twitter4j.Twitter;
import twitter4j.TwitterException;
import twitter4j.TwitterFactory;
import twitter4j.auth.AccessToken;
import twitter4j.conf.ConfigurationBuilder;
```

A FacebookClient object is created with the access token (given by user), and then that object is used to post the message. There are deprecation warning for “DefaultFacebookClient”, but I have found that it does not affect the functionality of the application in any way.



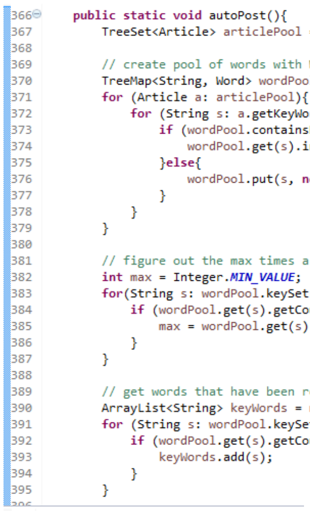
```java
    public boolean postToFb(String message){
        try{
            @SuppressWarnings("deprecation")
            FacebookClient client = new ~~DefaultFacebookClient~~(fbAccessToken);
            client.publish("me/feed", GraphResponse.class, Parameter.with("message", message));
            return true;
        }catch(FacebookException e){
            return false;
        }
    }
```

The postToTwitter(String tweet) method instantiates a ConfigurationBuilder object using the given consumer key, consumer secret, access token, and access secret, which is then used to build a Twitter instance. Posting is done by attempting (see the try/catch block) to update the user’s status to this tweet.

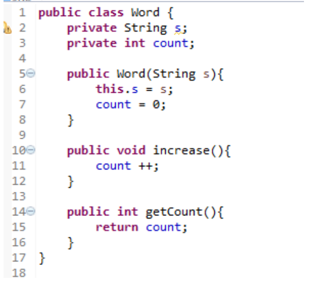


```java
    public boolean postToTwitter(String tweet){
        ConfigurationBuilder cb = new ConfigurationBuilder();
        cb.setDebugEnabled(true)
            .setOAuthConsumerKey(tConsumerKey)
            .setOAuthConsumerSecret(tConsumerSecret)
            .setOAuthAccessToken(tAccessToken)
            .setOAuthAccessTokenSecret(tAccessSecret);

        TwitterFactory tf = new TwitterFactory(cb.build());
        Twitter t = tf.getInstance();

        try {
            t.updateStatus(tweet);
            return true;
        } catch (TwitterException te) {
            te.printStackTrace();
            return false;
        }
    }
```

The autoPost() method in the Manager class takes the keywords from articles in myRecentNews, stores them in a TreeMap<String, Word> that maps a word to a Word object, which also stores the number of times the keyword shows up all the article’s keywords. This keyset of the wordPool TreeMap is iterated over, and the max count value is found. Then, the all the keywords that are repeated max times are added to the keyWords ArrayList.

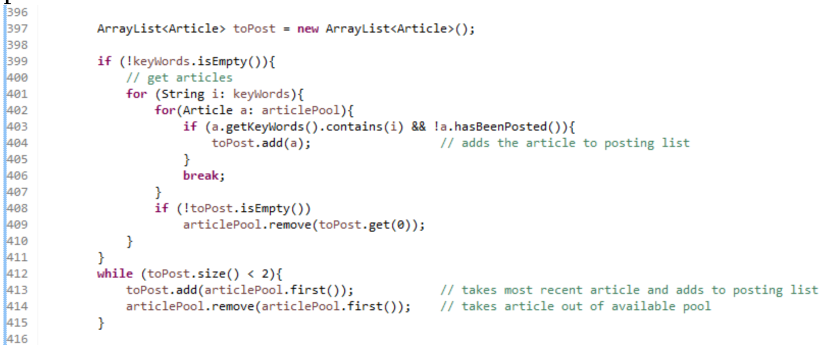


```java
    public static void autoPost(){
        TreeSet<Article> articlePool = c.myRecentNews;

        // create pool of words with Word object that stores and increases() its count
        TreeMap<String, Word> wordPool = new TreeMap<String, Word>();
        for (Article a: articlePool){
            for (String s: a.getKeyWords()){
                if (wordPool.containsKey(s)){
                    wordPool.get(s).increase();
                }else{
                    wordPool.put(s, new Word(s));
                }
            }
        }

        // figure out the max times a word is repeated
        int max = Integer.MIN_VALUE;
        for(String s: wordPool.keySet()){
            if (wordPool.get(s).getCount() > max){
                max = wordPool.get(s).getCount();
            }
        }

        // get words that have been repeated max times
        ArrayList<String> keyWords = new ArrayList<String>();
        for (String s: wordPool.keySet()){
            if (wordPool.get(s).getCount() == max){
                keyWords.add(s);
            }
        }
```



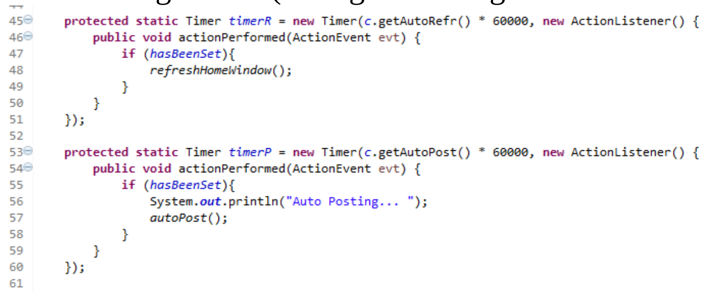
```java
public class Word {
    private String s;
    private int count;

    public Word(String s){
        this.s = s;
        count = 0;
    }

    public void increase(){
        count ++;
    }

    public int getCount(){
        return count;
    }
}
```

keyWords should not be empty, but if it is, the method skips over this block to the while(toPost.size() <2) block which fills the toPost ArrayList until it holds two articles with the most recent article in articlePool. Within the if statement, keyWords is iterated over (however it only holds one or two words), to find articles that contain that keyword and hasn't been posted yet, and is added to toPost and removed from the article pool.



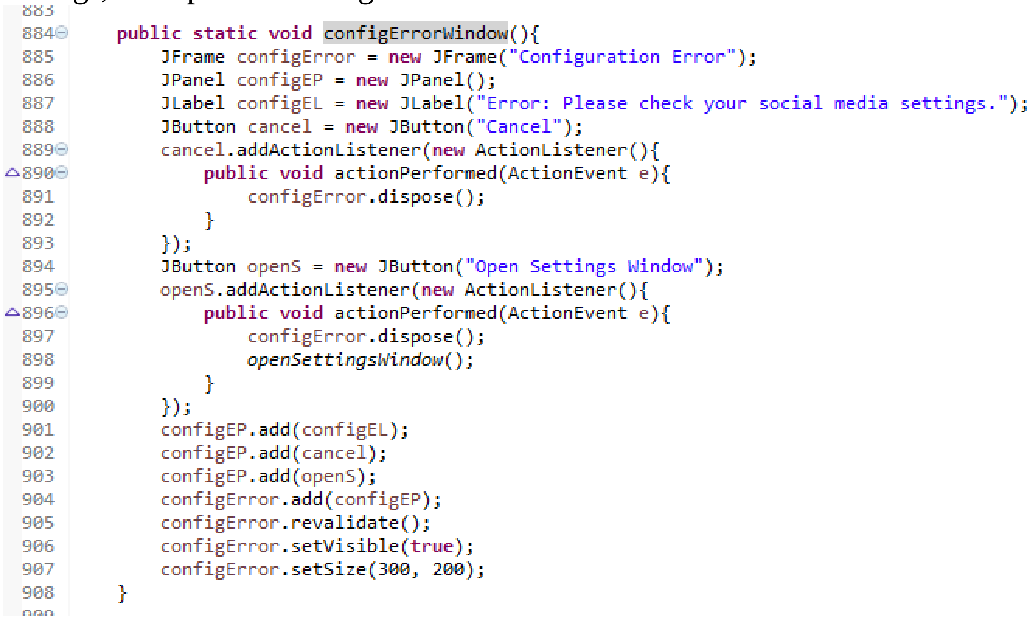
```java

        ArrayList<Article> toPost = new ArrayList<Article>();

        if (!keyWords.isEmpty()){
            // get articles
            for (String i: keyWords){
                for(Article a: articlePool){
                    if (a.getKeyWords().contains(i) && !a.hasBeenPosted()){
                        toPost.add(a);                  // adds the article to posting list
                    }
                    break;
                }
                if (!toPost.isEmpty())
                    articlePool.remove(toPost.get(0));
            }
        }
        while (toPost.size() < 2){
            toPost.add(articlePool.first());            // takes most recent article and adds to posting list
            articlePool.remove(articlePool.first());    // takes article out of available pool
        }

```

Timers play an important role in this application. The timers are set up within the Manager class, one for refreshing and another for auto-posting. Both are set to User c's settings (which are in minutes, while the Timer constructor takes milliseconds, hence there is the *60000 multiplier). Their ActionListener's are set to call the refreshHomeWindow() and autoPost() methods respectively if the user has progressed past the initial settings menu (saving the settings makes hasBeenSet true).



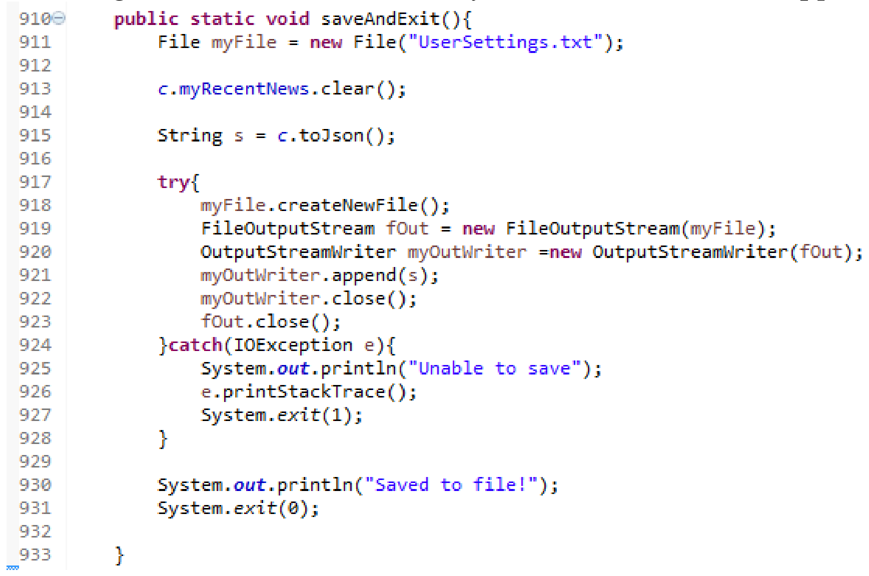
```java
    protected static Timer timerR = new Timer(c.getAutoRefr() * 60000, new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            if (hasBeenSet){
                refreshHomeWindow();
            }
        }
    });

    protected static Timer timerP = new Timer(c.getAutoPost() * 60000, new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            if (hasBeenSet){
                System.out.println("Auto Posting... ");
                autoPost();
            }
        }
    });

```

Here is a basic error window used to alert the user that their social media settings aren’t configured correctly. I include this in the product development documentation because it is one of the simplest GUI windows in this application and demonstrates how Java Swing is used. The settings window is more complicated, as it includes a JTabbedPane and pre-selects interests from c.getTopics() and frequency settings, and opens an configuration instruction window.



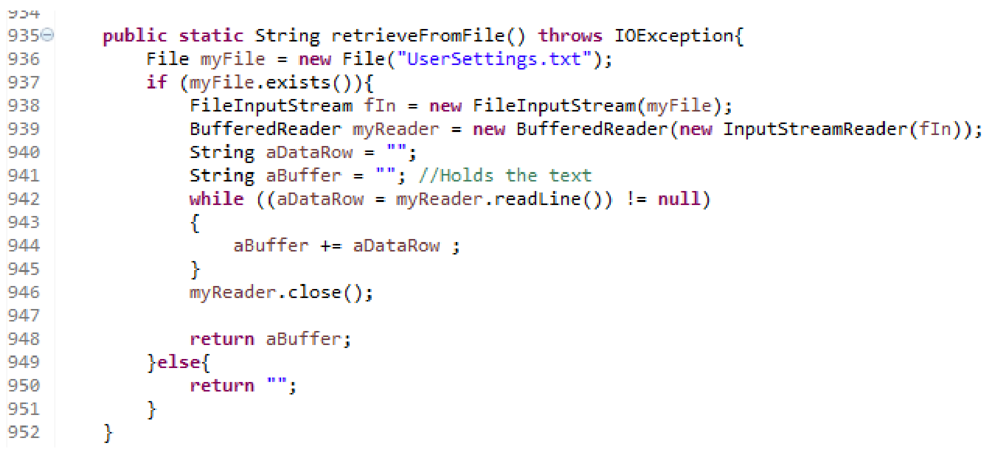
```
    public static void configErrorWindow(){
        JFrame configError = new JFrame("Configuration Error");
        JPanel configEP = new JPanel();
        JLabel configEL = new JLabel("Error: Please check your social media settings.");
        JButton cancel = new JButton("Cancel");
        cancel.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                configError.dispose();
            }
        });
        JButton openS = new JButton("Open Settings Window");
        openS.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                configError.dispose();
                openSettingsWindow();
            }
        });
        configEP.add(configEL);
        configEP.add(cancel);
        configEP.add(openS);
        configError.add(configEP);
        configError.revalidate();
        configError.setVisible(true);
        configError.setSize(300, 200);
    }
```

Retrieving and saving from a files is also an important feature of application, as it allows the user to not have to input all their information multiple times. Since I already used Gson to convert from the JSON NewsAPI response, it made sense to use it again to write to the file. Before converting the User object to JSON, c.myRecentNews is cleared out, as keeping them does not provide a significant advantage over refreshing, even if the articles aren’t yet out of date when the application is reopened.



```
    public static void saveAndExit(){
        File myFile = new File("UserSettings.txt");

        c.myRecentNews.clear();

        String s = c.toJson();

        try{
            myFile.createNewFile();
            FileOutputStream fOut = new FileOutputStream(myFile);
            OutputStreamWriter myOutWriter =new OutputStreamWriter(fOut);
            myOutWriter.append(s);
            myOutWriter.close();
            fOut.close();
        }catch(IOException e){
            System.out.println("Unable to save");
            e.printStackTrace();
            System.exit(1);
        }

        System.out.println("Saved to file!");
        System.exit(0);

    }
```

At startup, the main method of Manager calls this retrieveFromFile() method, which returns the contents of UserSettings.txt if it exists using FileInputStream, InputStreamReader, and BufferedReader: a blank String if this file does not exist. This string is in a JSON format, and is converted to the User object c in the main method. Otherwise, a name input window is opened first to create a new User object.



```java
public static String retrieveFromFile() throws IOException{
    File myFile = new File("UserSettings.txt");
    if (myFile.exists()){
        FileInputStream fIn = new FileInputStream(myFile);
        BufferedReader myReader = new BufferedReader(new InputStreamReader(fIn));
        String aDataRow = "";
        String aBuffer = ""; //Holds the text
        while ((aDataRow = myReader.readLine()) != null)
        {
            aBuffer += aDataRow ;
        }
        myReader.close();

        return aBuffer;
    }else{
        return "";
    }
}
```