

Application User Interfaces

Assignment 2

John Byrne / EVCOM3 / john.byrne@mycit.ie

Specification:

Design and implementation of a Twitter Client Mock-up integrating the following functionality...

- Login screen which validates credentials for user 'test' with password 'test'.
- Ability to Save and Load a Twitter Session
- Ability to Re-Tweet and mark selected tweets as favourites.
- Ability to write new tweets (up to a maximum of 140 characters.)
- Simulated 'live' twitter feed.
- Use of a Look and Feel theme to add a 'cool' factor.

Written by: John Byrne – R00050076

Due by: 18:00 on Wed 13th May 2015

1 TABLE OF CONTENTS

2	Assignment Requirements	4
3	List of Classes and Block Diagram	5
4	Walk through	7
4.1	Req 1(a): Basic application functionality with a JFrame, menu and menu-items.....	7
4.2	Req 1(b): 2 JTextFields and a (J)Button to login with name (test) and password (test)	9
4.3	Req 1(c): A (J)List to view incoming tweets and sent tweets.....	12
4.4	Req 1(d): A (J)Text Area for sending a new tweets and a Send button.	16
4.4.1	JTextArea and sending tweets... ..	16
4.4.2	Separate array for the hardcoded incoming tweets... ..	18
4.5	Req 1(e): Use of Colours & different Fonts.	20
4.6	Req 2(a): Re-tweeting received tweets. Allow marking some received tweets as favourite.	21
4.7	Req 2(b): Customized component extended from (J)Panel.....	23
4.8	Req 2(c): Save application data to disk e.g. account info, tweets etc.	26
4.9	Req 3(a): Commented code & the use a coding standard. Use extra classes.....	28
4.10	Req 3(b): This document, App Specification, List of classes & Evaluation.....	29
5	Evaluation and Conclusion	30
6	Appendix A - Complete Source Code	31
6.1	Main.java.....	31
6.2	LoginForm.java.....	32

6.3	TwitterClient.java.....	33
6.4	TweetCellRenderer.java.....	40
6.5	TweetCellTemplate.java.....	42
6.6	Tweet.java.....	43

2 ASSIGNMENT REQUIREMENTS

AUI Assignment Two - Twitter v1.0

<Name/EVCOM3/Email> ← fill in please & then pages 3+

- 1. Design a Java Application User Interface that acts as a Twitter client.** Note you are only required to write the user interface – no connection to Twitter is required. The application should include the following:

- (a) Basic application functionality with a JFrame, menu and menu-items. DONE
- (b) 2 JTextFields and a (J)Button to login with name (test) and password (test) to the twitter account. 3 attempts are allowed. A Dialog displays if more that 3 incorrect attempts are made. DONE
- (c) A (J)List to view incoming tweets and sent tweets. Choice or JCombobox button. DONE
- (d) A (J)Text Area for sending a new tweets and a Send button. Sent tweets are stored in an for e.g. array. Limit the characters to 140 for new tweets. Use another Array for hardcoded incoming tweets. DONE
- (e) Use of Colours & different Fonts. DONE

(10 marks ea, 50 marks total)

- 2. Three Advanced features or innovation/creativity; here are some examples *but you can do different ones if you want*:**

- (a) Re-tweeting received tweets. Allow marking some received tweets as favourite. DONE
- (b) Customized component extended from (J)Panel. DONE
- (c) Save application data to disk e.g. account info, tweets etc. DONE

(10 marks ea, 30 marks total)

- 3. Documentation includes the following:**

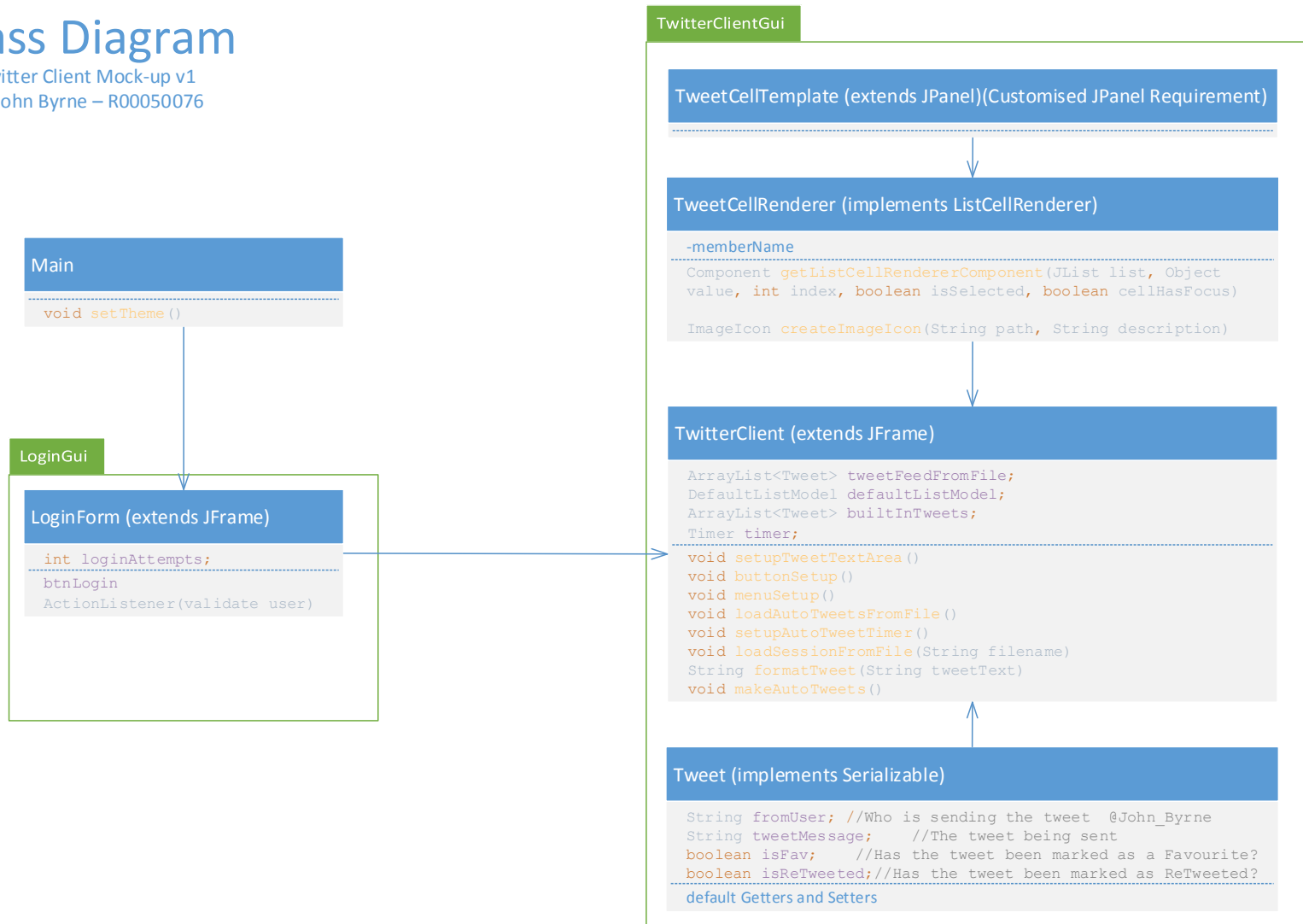
- a) Commented code & the use a coding standard. Use extra classes where appropriate. DONE
- b) This document stating what done including screen shots of code snippets & the running app code (on pages 3+). App Specification (~½ page). List of classes and what each is for. Evaluation (~½ page in 3 bullet points): how has it worked out, what could be done differently & future outlook. DONE

(10 marks ea, 20 marks total)

3 LIST OF CLASSES AND BLOCK DIAGRAM

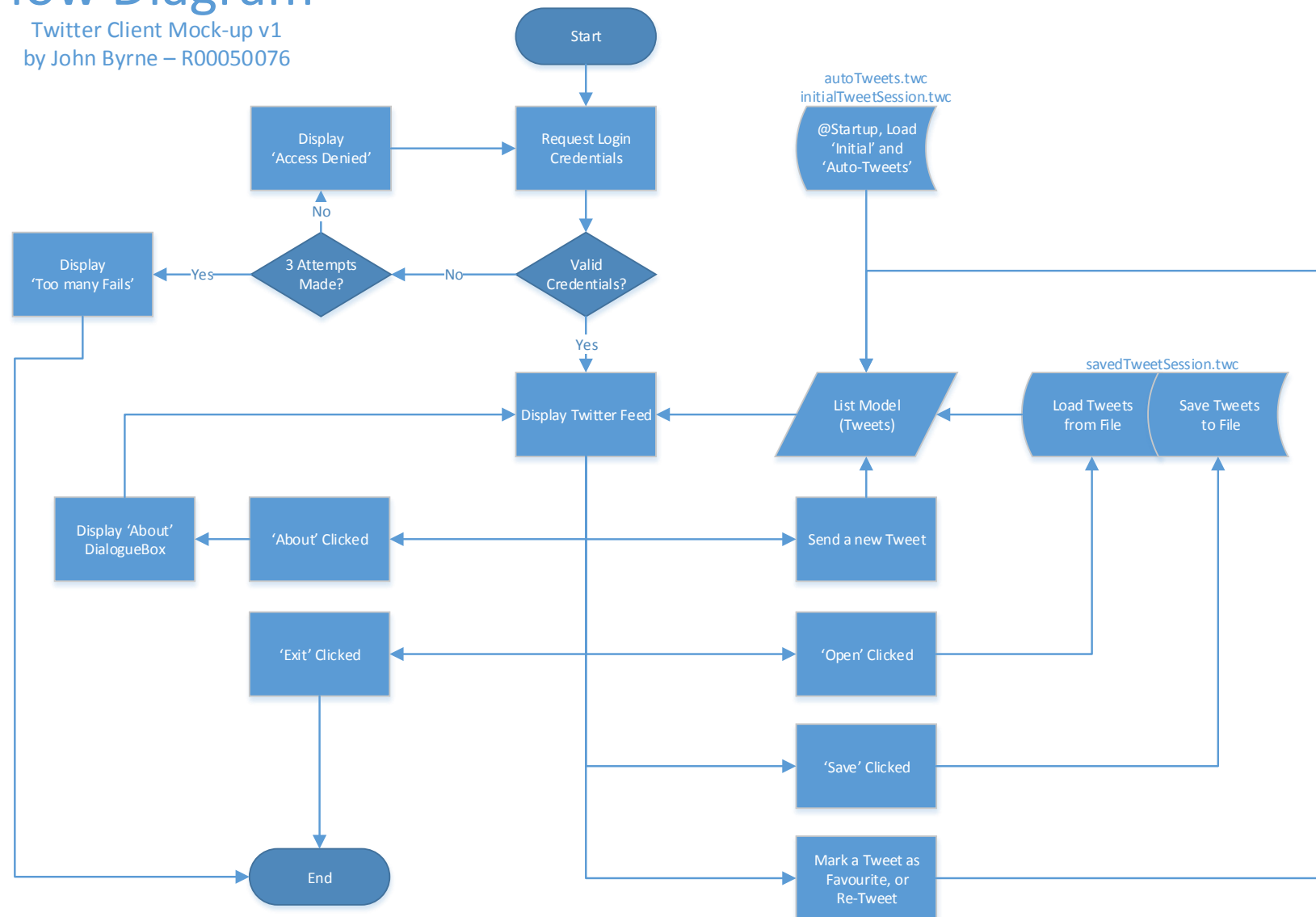
Class Diagram

Twitter Client Mock-up v1
by John Byrne – R00050076



Flow Diagram

Twitter Client Mock-up v1
by John Byrne – R00050076



4 WALK THROUGH

Below, I address each requirement to show how it has been addressed.

4.1 REQ 1(A): BASIC APPLICATION FUNCTIONALITY WITH A JFrame, MENU AND MENU-ITEMS.

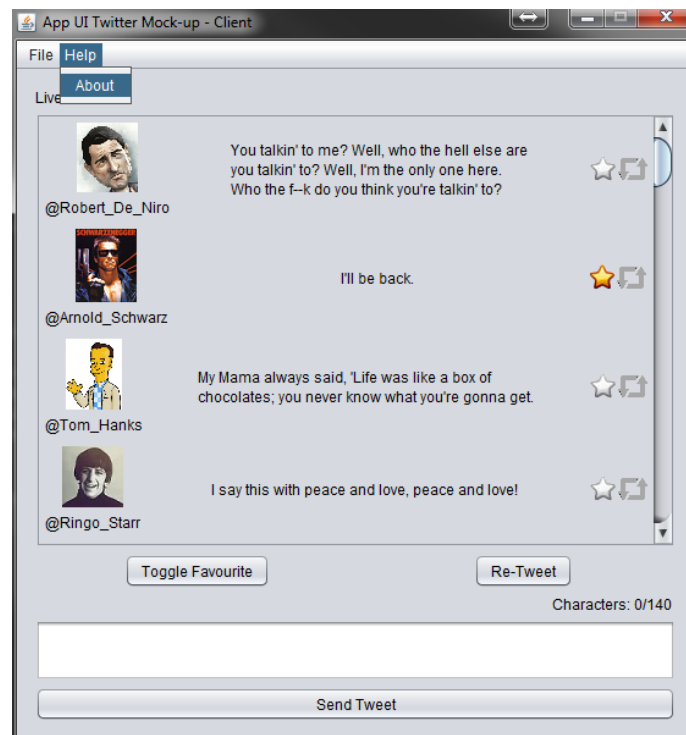


Fig 1. Client screenshot showing JFrame with Menu and MenuItems

Let's have a look at the essential code required to accomplish this...

```

/**
 * Application User Interfaces Assignment II
 * by John Byrne - R00050076.
 */
public class TwitterClient extends JFrame {
    private JPanel twitterClient;
    private JButton btnSendTweet;
    private JList lstTweets;
    private JTextArea txtTweet;
    private JButton btnFavourite;
    private JButton btnReTweet;
    private JLabel lblOf140Chars;

    private ArrayList<Tweet> tweetFeedFromFile; //Transition ArrayList. Store the tweets loaded from file,
                                                // before they are passed into the defaultListModel
    DefaultListModel defaultListModel; //The data model for the JList
    ArrayList<Tweet> builtInTweets; //An ArrayList storing a series of 'built-in tweets' to simulate a feed.
    Timer timer; //Used to spread-out when the 'built-in tweets' are added.

    public TwitterClient() {
        super("App UI Twitter Mock-up - Client");

        //// Initialise may variables
        tweetFeedFromFile = new ArrayList<>();

```

Fig 2. Class and Constructor definition for TwitterClient JFrame

```

196  /**
197   * This method sets up the Menu, MenuItems and their ActionListeners
198   */
199  private void menuSetup() {
200
201      ////////////////
202      /// Menu Items ///
203      ////////////////
204      JMenuBar menuBar = new JMenuBar();
205      setJMenuBar(menuBar);
206
207      JMenu file = new JMenu("File");
208      menuBar.add(file);
209      JMenuItem open = new JMenuItem("Open");
210      file.add(open);
211      JMenuItem save = new JMenuItem("Save");
212      file.add(save);
213      file.addSeparator();
214      JMenuItem exit = new JMenuItem("Exit");
215      file.add(exit);
216
217      JMenu help = new JMenu("Help");
218      menuBar.add(help);
219      JMenuItem about = new JMenuItem("About");
220      help.add(about);

```

Fig 3. The beginning of the *menuSetup()* method.

The above code shows the essential code required to achieve this requirement. We start off with our 'TwitterClient' JFrame which contains the code for the frame's content. The constructor of this JFrame calls the *menuSetup()* method, which sets-up and adds the JMenuBar and JMenuItem.

4.2 REQ 1(B): 2 JTEXTFIELDS AND A (J)BUTTON TO LOGIN WITH NAME (TEST) AND PASSWORD (TEST) TO THE TWITTER ACCOUNT. 3 ATTEMPTS ARE ALLOWED. A DIALOG DISPLAYS IF MORE THAN 3 INCORRECT ATTEMPTS ARE MADE.

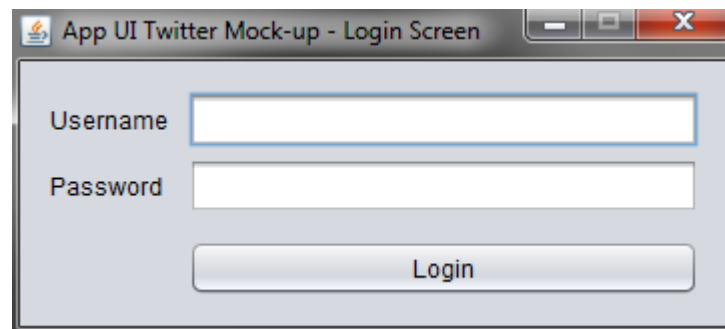


Fig 4. Login Window showing 2 x JTextFields and a JButton

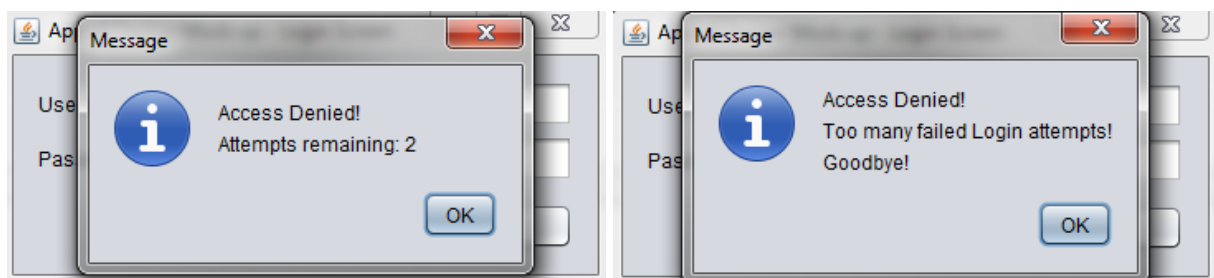


Fig 5. A series of Failed login attempts resulting in 'Denied!' Dialogues.

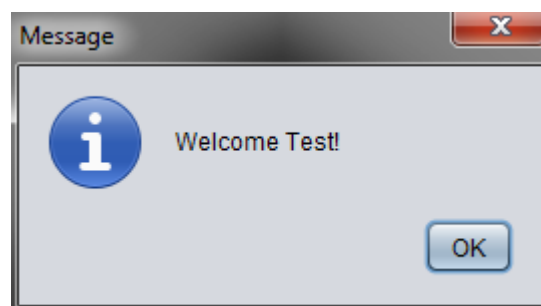


Fig 6. Login success when the correct credentials are given.

Let's have a look at the code required for this...

```

16 public class LoginForm extends JFrame{
17     private JPanel loginPanel;
18     private JButton btnLogin;
19     private JTextField txtUsername;
20     private JPasswordField txtPassword;
21     private int loginAttempts;           //Track the number of log-ins attempted in this session.
22
23     public LoginForm(){
24         super("App UI Twitter Mock-up - Login Screen");
25         setContentPane(loginPanel);
26         pack();
27         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28
29         //Initialise instance variable;
30         loginAttempts = 0;
31
32         /**
33          * Only one action is performed on this window; to validate the users' login attempt.
34          */
35         btnLogin.addActionListener(new ActionListener(){
36             @Override
37             public void actionPerformed(ActionEvent e) {

```

Fig 7. The LoginForm Class

We start off our code, defining the instance variables for this class. These are followed by the Constructor which adds an ActionListener to the only button, initialises, sizes and sets-Visible, the frame. All the logic for this frame is achieved within the button's ActionListener...

```

32     /**
33      * Only one action is performed on this window; to validate the users' login attempt.
34      */
35     btnLogin.addActionListener(new ActionListener(){
36         @Override
37         public void actionPerformed(ActionEvent e) {
38
39             loginAttempts++;
40             String username = txtUsername.getText();
41             String password = String.valueOf(txtPassword.getPassword());
42
43             if (username.equals("test") && password.equals("test")) {
44                 setVisible(false);
45                 JOptionPane.showMessageDialog(LoginForm.this, "Welcome " +
46                     WordUtils.capitalize(username) + "!");
47                 new TwitterClient();
48             } else {
49                 if (loginAttempts < 3) {
50                     JOptionPane.showMessageDialog(LoginForm.this,
51                         "Access Denied!\nAttempts remaining: " + (3 - loginAttempts));
52                 } else { //On third failed login attempt.
53                     JOptionPane.showMessageDialog(LoginForm.this,
54                         "Access Denied!\nToo many failed Login attempts!\nGoodbye!");
55                     System.exit(0);
56                 }
57             }
58         }
59     }

```

Fig 8. The 'Login' button, ActionListener.

When the button is clicked, we count this as a login attempt, so on line 39 we increment the *loginAttempts* variable by 1.

Line 40 and 41 extract the user input from the input fields on the JFrame. The password field returns an Array of Chars, so I convert this to a String for ease of comparison.

Line 43 begins our conditional operation, where the user input is compared against the Strings 'test' and 'test'. If we have a match, a welcome dialogue is displayed on line 45 and the TwitterClient GUI is instantiated on 47. If on the other hand we don't have a match, then on line 50 we check to see if this is the users 3rd attempt to login. If not, then on 51 we simply tell the user 'Access Denied' and warn them of the number of attempts they have remaining. If this was their 3rd attempt, then line 55 shows a dialogue informing the user that they have made too many failed attempts and the program exits on line 57.

4.3 REQ 1(c): A (J)LIST TO VIEW INCOMING TWEETS AND SENT TWEETS.

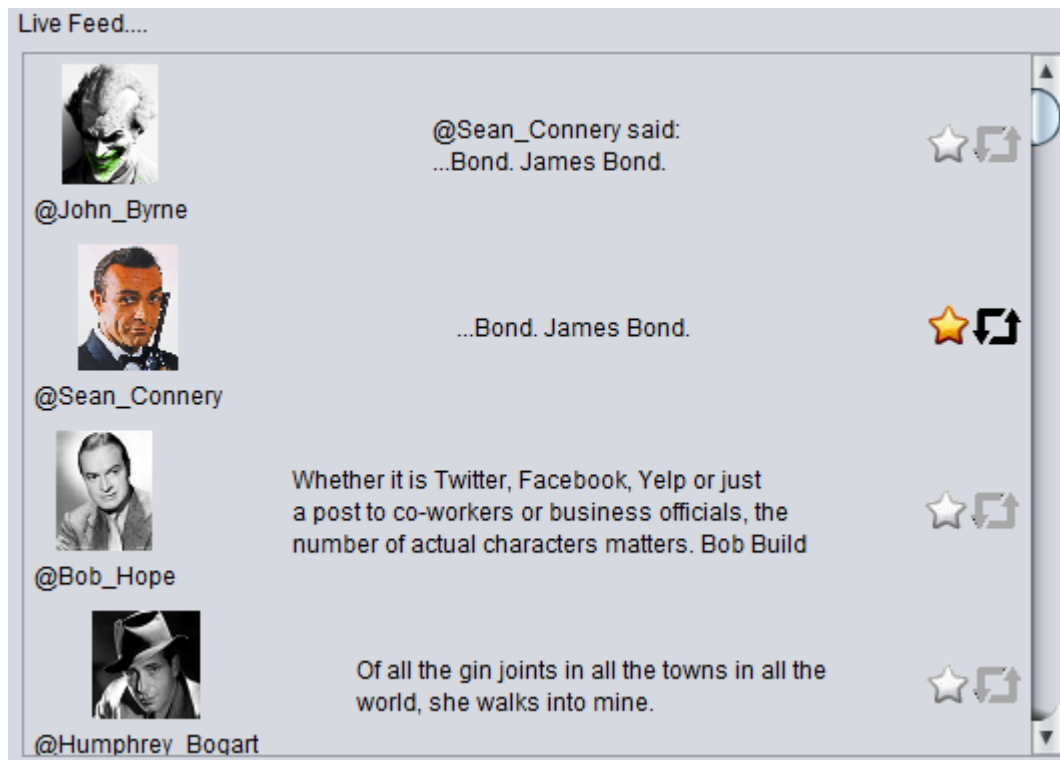


Fig 9. The JList part of the TwitterClient JFrame, showing several Tweets.

The code responsible for delivering what you see above, involves several classes. Let's start with the TwitterClient class which ultimately displays our list to the user...

```

31  public TwitterClient() {
32      super("App UI Twitter Mock-up - Client");
33
34      //// Initialise may variables
35      tweetFeedFromFile = new ArrayList<>();
36      defaultListModel = new DefaultListModel();
37      builtInTweets = new ArrayList<>();
38
39      //// Configure the JFrame
40      setContentPane(twitterClient);
41      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
42
43      //// JList Formatting...How the Live Feed looks and feels...
44      lstTweets.setCellRenderer(new TweetCellRenderer());
45      lstTweets.setModel(defaultListModel);
46
47      //// Read in the auto Tweets from file

```

Fig 10. A section of the TwitterClient constructor.

Within the constructor of my TwitterClient class, on line 44 I set the JList's cell renderer to

TweetCellRenderer, a custom class which implements ListCellRenderer. The ListCellRenderer will

return a Component to be used by the JList as an element. I achieve a customised design for my Twitter feed by governing the design of the component being returned by the ListCellRenderer. In this project, the ListCellRenderer instantiates a customised JPanel called a TweetCellTemplate, which is populated and adjusted based on the attributes of the Tweet to be displayed. Once the TweetCellTemplate JPanel is ready, it is passed back to the JList as the component to be used for the list's element. **See Req 2(b) below for development details of the TweetCellTemplate class.** In the meantime, we'll take a closer look at the TweetCellRenderer.

```

7  /**
8   * My Tweet Cell Renderer class
9   * Each cell in my JList is rendered based on the rules defined here.
10  * by John Byrne - R00050076
11  */
12  public class TweetCellRenderer implements ListCellRenderer {
13
14      @Override
15      public Component getListCellRendererComponent(JList list, Object value, int index,
16                                                    boolean isSelected, boolean cellHasFocus) {

```

Fig 11. TweetCellRendered class definition which implements ListCellRenderer, with mandatory method.

Implementing the ListCellRenderer requires a concrete implementation of the *getListCellRendererComponent* method which has the following parameters...

- JList list** This is the JList associated with this Renderer class. As in our *IstTweets* JList.
- Object value** This is the Object whose data the JList is trying to represent. These are Tweets.
- int index** The index of this element in the JList
- boolean isSelected** Has this element been selected? True or False
- boolean callHasFocus** Does this cell have focus? True or False

The TweetCellRenderer acts like an ActionListener and its' method acts like the actionPerformed method. These parameters do not need to be passed to this method explicitly, but rather they are already populated by the JList when calling on its' cellRenderer. This means that these parameters are already populated within this method. We need to process these to determine the correct actions and behaviours.

For the purposes of this project, we don't need to do anything specifically with the 'JList list' parameter.

The 'Object value' parameter comes from our data model, each element of which, the JList is trying to represent. Since the data model is made up of Tweet objects, we can cast this value as a Tweet...

```

21      //Get the data and cast it as a Tweet object
22      Tweet tweet = (Tweet)value;
23      /* From Tweet object
24      this.fromUser = fromUser;
25      this.tweetMessage = tweetMessage;
26      this.isFav = isFav;
27      this.isReTweeted = isReTweeted;
28      */

```

Fig 12. Object value being cast as a Tweet

From the Tweet instance, we can access and extract its' inherent data...

```

30      //Extract all the data from the Tweet...
31      String fromUser = tweet.getFromUser();
32      String tweetMessage = tweet.getTweetMessage();
33      boolean isFav = tweet.isFav();
34      boolean isReTweeted = tweet.isReTweeted();

```

Fig 13. Extracting data from our Tweet instance.

We can then instantiate a new TweetCellTemplate and update the attributes of this instance with the data extracted from our Tweet instance...

```

36      //Build the Tweet cell...
37      TweetCellTemplate cell = new TweetCellTemplate(); //Create an instance of my Cell template, which I will
38      //ultimately pass back to the JList as an element.
39      ImageIcon icon = createImageIcon("../Images/" + fromUser + ".png", "User Profile Picture");
40      //Get profile picture
41      cell.getLblProfilePic().setIcon(icon); //Set the profile picture
42      cell.getLblProfilePic().setText(fromUser); //Set the profile name
43      cell.getLblTweet().setText(tweetMessage); //Set the Tweet message
44
45      //The cells default to ifFav and isReTweeted == false, so is I set isFav to true and back again to false,
46      //the cell will return to default configuration, so I don't need to repaint the isNotFav icon at runtime.
47      if(isFav){
48          ImageIcon favIcon = createImageIcon("../Images/isFavStar.png", "a Favourite Tweet");
49          cell.getLblFav().setIcon(favIcon);
50      }
51      if(isReTweeted){
52          ImageIcon reTweetIcon = createImageIcon("../Images/isReTweeted.png", "a ReTweeted Tweet");
53          cell.getLblReTweet().setIcon(reTweetIcon);
54      }

```

Fig 14. Instantiation of TweetCellTemplate object and updating of its' attributes

In this project, I don't make use of the *isSelected* parameter. Instead I use the *cellHasFocus* parameter to determine if this item in the list has been clicked or not and then return the cell.

```
60         if(cellHasFocus){
61             //Color origCol = cell.getBackground();
62             //cell.getLabelReTweet().setText(origCol.toString());
63             cell.setBackground(cellSelectedColor);
64         }
65
66
67         return cell;
68     }
```

Fig 15. If a cell has focus, its' background colour should change to represent this fact to the user.

4.4 REQ 1(D): A (J)TEXT AREA FOR SENDING A NEW TWEETS AND A SEND BUTTON. SENT TWEETS ARE STORED IN AN FOR E.G. ARRAY. LIMIT THE CHARACTERS TO 140 FOR NEW TWEETS. USE ANOTHER ARRAY FOR HARDCODED INCOMING TWEETS.

4.4.1 JTextArea and sending tweets...

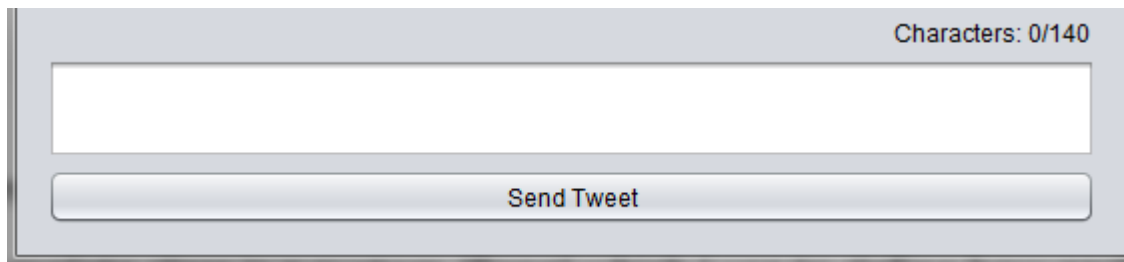


Fig 16. The JTextArea for sending new tweets and a 'Send Tweet' button.

The JTextArea is part of the TwitterClient. Part of the TwitterClient constructor, calls the *setupTweetTextArea()* method which adds a DocumentListener to the JTextArea which I use to control the character counter, showing the user how many characters have been typed in their pending tweet...

```

83  /**
84   * The method adds a DocumentListener to the TextArea.
85   * The Document listener acts upon changes to the textArea and updates the character counter.
86   */
87  private void setupTweetTextArea() {
88      ///////////////////////////////////////////////////
89      // Tweet textarea configuration //
90      ///////////////////////////////////////////////////
91      txtTweet.getDocument().addDocumentListener(new DocumentListener() {
92          @Override
93          public void insertUpdate(DocumentEvent e) {
94              lblOf140Chars.setText(String.valueOf("Characters: " + e.getDocument().getLength() + "/140"));
95              if (e.getDocument().getLength() > 140) {
96                  lblOf140Chars.setForeground(Color.RED);
97              }
98              //Characters: 0 out of a maximum 140
99          }
100
101          @Override
102          public void removeUpdate(DocumentEvent e) {
103              lblOf140Chars.setText(String.valueOf("Characters: " + e.getDocument().getLength() + "/140"));
104              if (e.getDocument().getLength() <= 140) {
105                  lblOf140Chars.setForeground(Color.BLACK);
106              }
107          }
108
109          @Override
110          public void changedUpdate(DocumentEvent e) {
111              //
112          }
113      });
114  }
115  }

```

Fig 17. The *setupTweetTextArea()* code, which adds a DocumentListener

The logic is quite straight forward. Whether content is being added or removed from the JTextArea, we first update the JLabel which informs the user of the number of characters in the field. If the number of characters is greater than 140, then this JLabel should display RED text, or otherwise remain BLACK if there are less than 140 characters. The actual control of whether or not the typed tweet is acceptable or not, is done within the ActionListener of the 'Send Tweet' button...

```

172      /**
173       * Send Tweet Button Action Listener
174       * If the message is not too long, it takes the content
175       * of the Tweet TextArea and add a new Tweet to the feed.
176       */
177      btnSendTweet.addActionListener(new ActionListener() {
178          @Override
179          public void actionPerformed(ActionEvent e) {
180              if (txtTweet.getDocument().getLength() > 140) {
181                  JOptionPane.showMessageDialog(TwitterClient.this,
182                      "Your tweet is too long!\n Maximum of 140 characters.");
183              } else if (txtTweet.getDocument().getLength() > 0) {
184                  String formattedTweet = formatTweet(txtTweet.getText());
185                  defaultListModel.add(0, new Tweet("@John_Byrne", formattedTweet, false, false));
186                  txtTweet.setText(""); //Blank out the new tweet text box, ready for a new entry.
187              }
188          }
189      });
190  }
191  }
192  }
193  }
194  }

```

Fig 18. The 'Send Tweet' button ActionListener

When the button is clicked, on line 180 we first check to see if the proposed tweet is not too long. If it is, on 181 we show a Message Dialogue to let the user know, just in case the RED character counter wasn't enough for them! If the message is an acceptable length, then on line 185 I pass the content of the JTextArea to the *formatTweet* method, which returns the message as a HTML string including
 tags where appropriate. This is done to ensure that the tweet fits in its' cell. On line 187 I instantiate a new Tweet with the formatted text and add it to the top of my JLists' data model. Lastly, on 189, I blank out the JTextArea in preparation for the next tweet.

4.4.2 Separate array for the hardcoded incoming tweets...



Fig 19. A sample of Incoming, Hard-coded Tweets

There are two primary methods which govern the Hard-coded tweets, or auto-tweets as I like to call them. The methods *loadAutoTweetsFromFile()* and *setupAutoTweetTimer()*, work within the TwitterClient...

```


317  /**
318   * This method loads the Auto-Tweets in the to Twitter Client GUI.
319   * This method is only called at start-up
320   */
321  private void loadAutoTweetsFromFile() {
322      FileInputStream fis = null;
323      try {
324          fis = new FileInputStream("autoTweets.twc");
325          ObjectInputStream ois = new ObjectInputStream(fis);
326          builtInTweets = (ArrayList<Tweet>) ois.readObject();
327          ois.close();
328      } catch (IOException ex) {
329          ex.printStackTrace();
330          //Could throw in a Dialogue box to say there was an issue, but I don't think it's necessary for this app.
331      } catch (ClassNotFoundException ex) {
332          ex.printStackTrace();
333      }
334  }
335  }

```

Fig 20. The *loadAutoTweetsFromFile()* method which populates the *builtInTweets* ArrayList.

The *loadAutoTweetsFromFile()* method does exactly what is says on the tin. A file called 'autoTweets.twc' is read-in and it contains a pre-created ArrayList of Tweets which are saved to the TwitterClients' *builtInTweets* ArrayList.

Now we need to simulate incoming tweets...



```

337  /**
338   * Starts a background timer that will add a random Auto-Tweet at a random interval, between 10 and 25 seconds.
339   */
340  private void setupAutoTweetTimer() {
341      Random random = new Random();
342      timer = new Timer(random.nextInt(15000) + 10000,
343                      new ActionListener() { //Every 10 - 25 seconds, do this...
344                          @Override
345                          public void actionPerformed(ActionEvent e) {
346                              defaultListModel.add(0, builtInTweets.get(random.nextInt(builtInTweets.size())));
347                          }
348                      });
349  }
350  }
351  timer.start();
352  }

```

Fig 21. The *setupAutoTweetTimer()* method which acts to simulate incoming tweets.

The *setupAutoTweetTimer()* method instantiates a Timer object which triggers an ActionListener at a random interval between 15 and 25 seconds. Within the ActionListener, on line 347, a Random tweet instance is picked from the *builtInTweets* ArrayList and this is added to the top of the JLists' data model.

4.5 REQ 1(E): USE OF COLOURS & DIFFERENT FONTS.

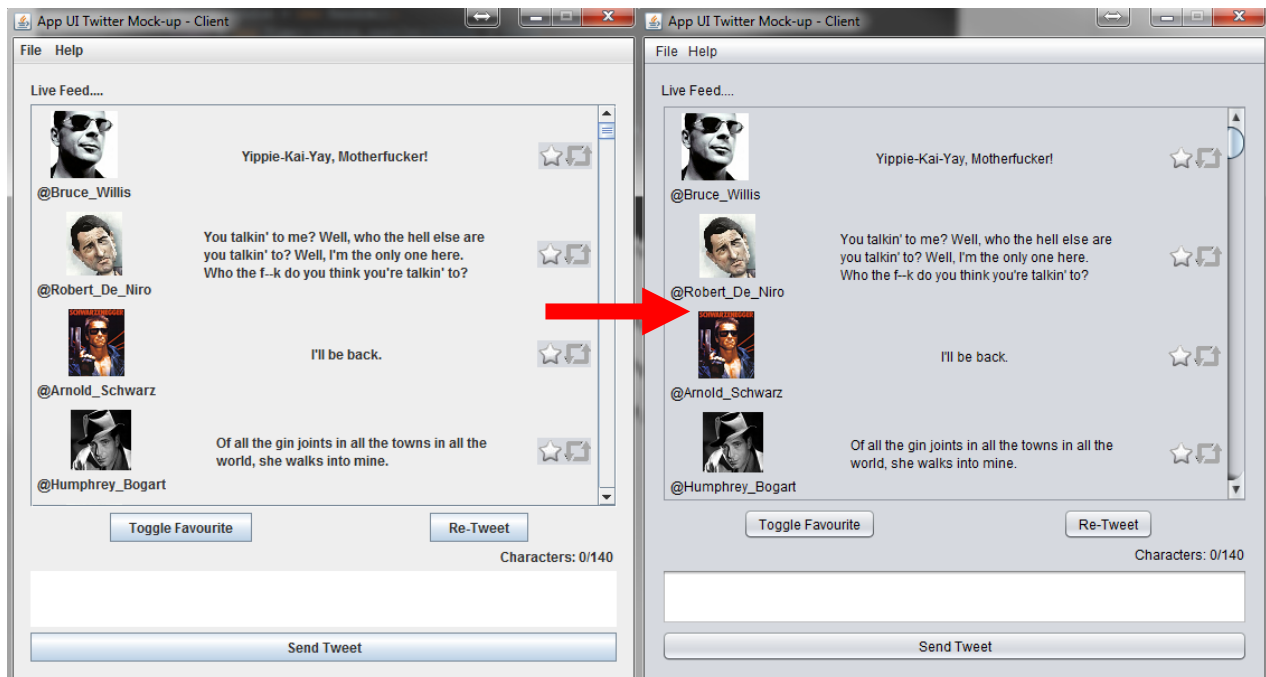


Fig 22. A comparison between my systems default theme on the Left and the Nimbus theme on the Right which I ultimately selected.

I have to straight up confess not to be a graphic designer and so I opted to evaluate the couple of Look and Feel themes supplied with my system by default. I found a theme by the name of Nimbus to be pleasant on the eye. With rounded buttons and scroll bar, a recessed JTextArea and a colour that while different to the standard windows theme, was similar enough not to be off putting. The code required to apply this is part of the Main.java file so that it would apply to all instantiated frames...

```

5  /**
6   * Application User Interfaces Assignment II
7   * by John Byrne - R00050076
8   *
9   * Twitter Client Mock-up Application Launcher
10  */
11  public class Main {
12      public static void main(String[] args) {...}
13
14      private static void setTheme() {
15          try {
16              for (UIManager.LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
17                  if ("Nimbus".equals(info.getName())) {
18                      UIManager.setLookAndFeel(info.getClassName());
19                      break;
20                  }
21              }
22          }
23      }
24  }

```

Fig 23. The Main.java file containing the `setTheme()` method.

The `setTheme()` method, checks each of the installed Look and Feel themes for one that has the name 'Nimbus'. If found, it requests the `UIManager` to set the Look and Feel to this theme. If this fails for whatever reason, the system's default theme will be used.

4.6 REQ 2(A): RE-TWEETING RECEIVED TWEETS. ALLOW MARKING SOME RECEIVED TWEETS AS FAVOURITE.

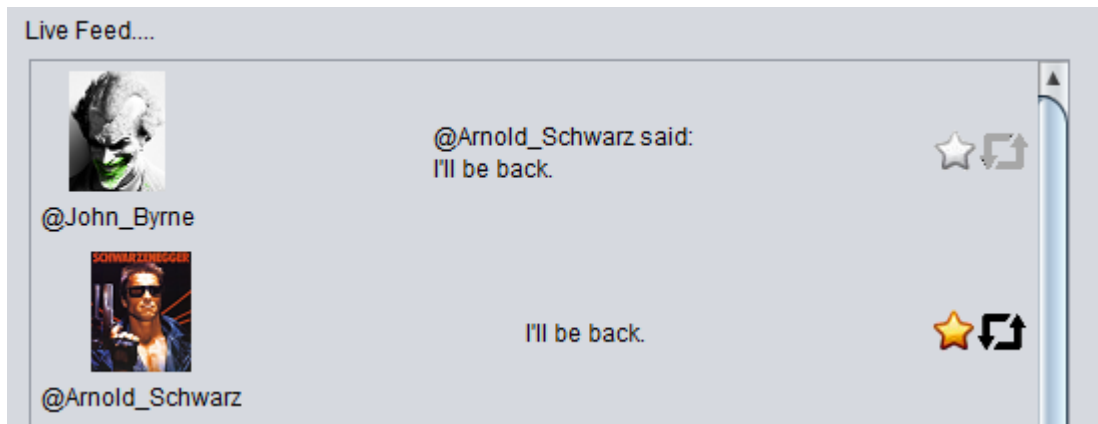


Fig 24. The tweet feed showing @Arnold_Schwarz's tweet being marked as a favourite and re-tweeted.

The code that governs this behaviour is contained primarily within the ActionListeners of the 'Toggle Favourite' and 'Re-Tweet' buttons...

```

122      /**
123       * Favourite Button Action Listener
124       * Sets the isFav flag of the selected Twest(s)
125       * The TweetCellRenderer looks after updating the Favourite icon once repaint() is called.
126       */
127      btnFavourite.addActionListener(new ActionListener() {
128          @Override
129          public void actionPerformed(ActionEvent e) {
130              int[] selectedItems = lstTweets.getSelectedIndices();
131              for (int item : selectedItems) {
132                  Tweet tweet = (Tweet)defaultListModel.get(item);
133                  if(tweet.isFav()){
134                      tweet.setIsFav(false);
135                  }else {
136                      tweet.setIsFav(true);
137                  }
138                  repaint();
139              }
140          }
141      });
142
143
144

```

Fig 25. The 'Toggle Favourite' button ActionListener.

Since each element in the JList is a ListCellRenderer representation of the Tweet instances in the JList's data model, if an element in the JList is selected, then the index of that element will be in the same position on the JList's data model. Updating the Tweet instance in the data model, will update its' representation in the JList. On line 130, I create an array of Selected Indices from the JList. While

this software currently only supports the selecting of one item, I use the `getSelectedIndices()` rather than the `getSelectedIndex()` method to allow for future functionality. On Line 133, I check to see if the tweet has already been marked as a Favourite or not and lines 134 and 136 toggle it one way or the other.

```

146      /**
147       * Re-Tweet Button Action Listener
148       * Sets the isReTweeted flag of the selected Tweet.
149       * The TweetCellRenderer looks after updating the Re-Tweeted
150       * icon once the re-tweeted tweet is displayed.
151       */
152      btnReTweet.addActionListener(new ActionListener() {
153          @Override
154          public void actionPerformed(ActionEvent e) {
155              int[] selectedItems = lstTweets.getSelectedIndices();
156              if (selectedItems.length > 1) {
157                  JOptionPane.showMessageDialog(TwitterClient.this,
158                      "Please select only one Tweet to re-Tweet. Tweet, Tweet! :)");
159              } else if (selectedItems.length > 0) {
160                  for (int item : selectedItems) {
161                      Tweet tweet = (Tweet) defaultListModel.get(item);
162                      if (!tweet.isReTweeted()) {
163                          tweet.setIsReTweeted(true);
164                      }
165                      defaultListModel.add(0, new Tweet("@John_Byrne", "<html>" + tweet.getFromUser() +
166                          " said: <br>" + tweet.getTweetMessage(), false, false));
167                  }
168              }
169          }
170      });

```

Fig 26. The 'Re-Tweet' button ActionListener.

The 'Re-Tweet' ActionListener logic works very similarly to the 'Toggle Favourite' ActionListener, in that we get the Selected Indices on line 155 and set the *isReTweeted* flag in the corresponding tweet instance as stored in the JList's data model on line 163. There are two main differences. Firstly, while it might be hypothetical to allow multiple selection when marking tweets as favourites, I do not want that behaviour for re-tweeting. Line 157 will display a Message Dialogue in the event that more than one tweet is selected. The second difference is that I need to create a new tweet containing the message from the tweet that I want to re-tweet, tweet 😊. This is done on line 165, where my new tweet is instantiated and added to the top of my JList data model.

4.7 REQ 2(B): CUSTOMIZED COMPONENT EXTENDED FROM (J)PANEL

The TweetCellTemplate class is a custom JPanel which I use to define how each tweet is displayed in the live feed. I used the GUI Form Designer of the IntelliJ IDE to generalise how I wanted the cell to look.

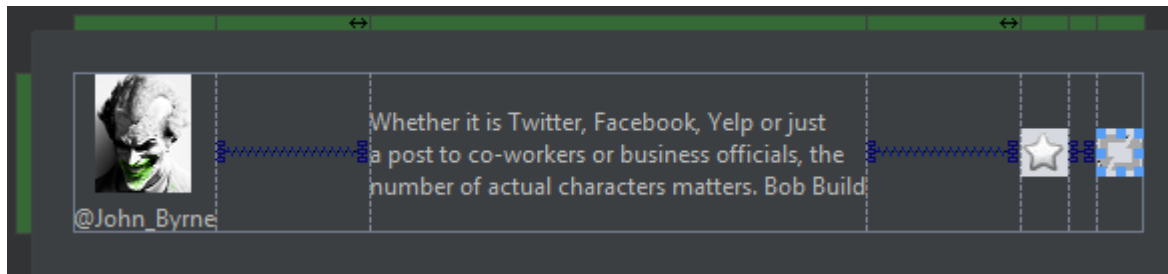


Fig 27. A prototype cell as designed in the GUI Form Designer.

I laid out my panel with a JLabel placed on the far left, in which I'll use to show the tweet source user and their profile picture.

In the middle, I want another JLabel showing the tweet message. This area needs to support a message of 140 characters and so needs to be split over multiple lines. A JLabel does not ordinarily support multi-lined input, but they do support HTML and so a String formatted as HTML with a few well-placed
 tags should allow this JLabel to accommodate the full message.

To the right of my custom JPanel, I want some icons representing if a tweet has been marked as a Favourite or if it has been Re-tweeted.

Once I had decided on the general look of the panel, I needed to code it...

```

7  /**
8   * This TweetCellTemplate class is a customised JPanel for use in my JList.
9   * by John Byrne - R00050076
10  */
11  public class TweetCellTemplate extends JPanel {
12      private JPanel rootTweet;
13      private JLabel lblProfilePic;
14      private JLabel lblTweet;
15      private JLabel lblFav;
16      private JLabel lblReTweet;
17
18      /**
19       * This constructor builds the JList Element.
20       */
21      public TweetCellTemplate() {
22          setBorder(new EmptyBorder(5, 5, 5, 5));
23          setLayout(new GridBagLayout());
24          GridBagConstraints gridBagConstraints = new GridBagConstraints();
25
26          gridBagConstraints.anchor = GridBagConstraints.WEST;
27          gridBagConstraints.weightx = 1;
28          add(lblProfilePic, gridBagConstraints);
29
30          add(lblTweet); //added without gridBagConstraints so that it will float
31
32          gridBagConstraints.anchor = GridBagConstraints.EAST;
33          add(lblFav, gridBagConstraints);
34          add(lblReTweet);
35      }

```

Fig 28. My customised JPanel which governs how each tweet will be displayed on screen.

I used a GridBagLayout manager and selectively applied constraints to some components to have them stick Left or Right, or remain floating. The components are added in order from Left to right.

Line 26 above, sets my first constraint, that being to anchor WEST (Left to you and me!). I then apply a 'weight' which is like a proportion or priority rating. Because I don't specify any competing weight, this is interpreted to mean that the component should consume all space. On line 28, I apply these two constraints to the *lblProfilePic* JLabel component, which consumes all the space and sits tight against the left side.

On line 30, I add the *lblTweet* JLabel which will display the tweet message. I don't apply any constraints to this component and so it will float in the middle of the JPanel and shrink or grow to accommodate its' contents.

On line 32, I change my anchor constraint from WEST to EAST (that's Right to you and me!). The 'weight' defined earlier is still in effect, so no need to re-specify this constraint. On line 33, I apply these two constraints to the *lbIFav* JLabel which consumes all available space and sticks to the Right side.

But then on line 34, I add my final JLabel, *lbReTweet* which is wedged in-between the *lbIFav* and the Right side, thus completing the layout of my custom JPanel.

4.8 REQ 2(c): SAVE APPLICATION DATA TO DISK E.G. ACCOUNT INFO, TWEETS ETC.

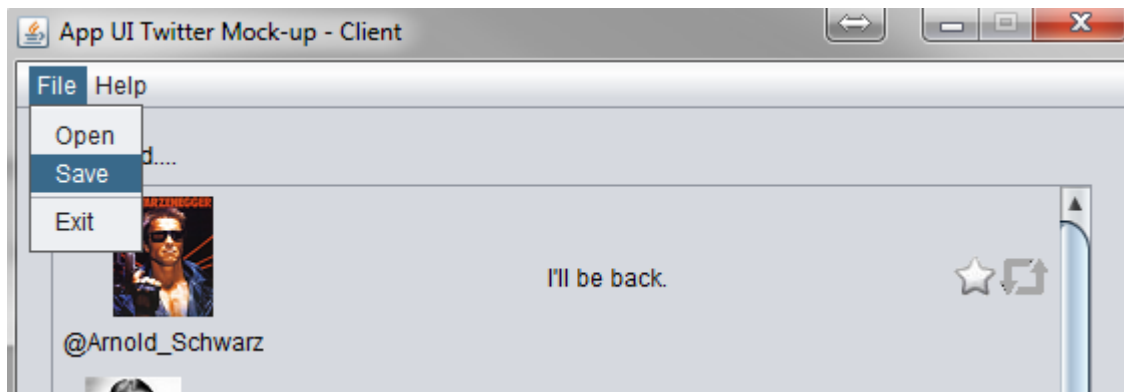


Fig 29. The 'Save' MenuItem of my TwitterClient

The save operation is handled by the 'Save' MenuItem's ActionListener...

```

276  /**
277   * Save MenuItem Action Listener.
278   * It produces an Option Dialogue box to get confirmation.
279   * If confirmed, it saves the twitter feed to the "savedTweetSession.twc" file.
280   */
281  save.addActionListener(new ActionListener() {
282      @Override
283      public void actionPerformed(ActionEvent e) {
284
285          //Create an OK / Cancel Dialogue box
286          Object[] options = {"OK", "Cancel"};
287          int overwriteConfirmation = JOptionPane.showOptionDialog(TwitterClient.this,
288              "*** Warning ** Previously saved session will be overwritten. Continue?",
289              "Confirm overwrite session ",
290              JOptionPane.PLAIN_MESSAGE,
291              JOptionPane.QUESTION_MESSAGE,
292              null,
293              options,
294              options[0]);
295
296          if (overwriteConfirmation == 0) {
297              ArrayList<Tweet> sessionTweets = new ArrayList<>();
298
299              for (int index = 0; index < defaultListModel.getSize(); index++) {
300                  Tweet tweet = (Tweet) defaultListModel.get(index);
301                  sessionTweets.add(tweet);
302              }
303
304              try {
305                  FileOutputStream fos = new FileOutputStream("savedTweetSession.twc");
306                  ObjectOutputStream oos = new ObjectOutputStream(fos);
307                  oos.writeObject(sessionTweets);
308                  oos.close();
309              } catch (IOException ex) {
310                  ex.printStackTrace();
311              }
312          }
313      }
314  });
315  }

```

Fig 30. The 'Save' MenuItem's ActionListener.

If the 'Save' MenuItem has been clicked on, we start on line 287 by generating a Dialogue to get confirmation from the user. This is done because if the user chooses to proceed, they will over write the pre-existing file. Line 296 checks for this confirmation.

What we actually save is an ArrayList of Serialized Tweet Instances. To generate this, on line 299 I iterate through the JList's data model and add each tweet to a local *sessionTweets* ArrayList on line 301. Once complete, from line 304 I try to write the data out to the 'savedTweetSession.twc' file.

4.9 REQ 3(A): COMMENTED CODE & THE USE A CODING STANDARD. USE EXTRA CLASSES WHERE APPROPRIATE.

My code is commented fairly extensively throughout my project, with each method and class having a JavaDoc style description and brief, while each custom variable also has a brief description of what it's used for...

```

25     private ArrayList<Tweet> tweetFeedFromFile; //Transition ArrayList. Store the tweets loaded from file,
26                                           // before they are passed into the defaultListModel
27     DefaultListModel defaultListModel;        //The data model for the JList
28     ArrayList<Tweet> builtInTweets;           //An ArrayList storing a series of 'built-in tweets' to simulate a feed.
29     Timer timer;                             //Used to spread-out when the 'built-in tweets' are added.

```

Fig 31. Examples of commented variables.

```

83     /**
84      * The method adds a DocumentListener to the TextArea.
85      * The Document listener acts upon changes to the textArea and updates the character counter.
86      */
87     private void setupTweetTextArea() {...}
116
117     /**
118      * This method sets up the Buttons and their ActionListeners
119      */
120     private void buttonSetup() {...}
195
196     /**
197      * This method sets up the Menu, MenuItems and their ActionListeners
198      */
199     private void menuSetup() {...}
316
317     /**
318      * This method loads the Auto-Tweets in the to Twitter Client GUI.
319      * This method is only called at start-up
320      */
321     private void loadAutoTweetsFromFile() {...}

```

Fig 32. Examples of commented methods.

```

7     /**
8      * My Tweet Cell Renderer class
9      * Each cell in my JList is rendered based on the rules defined here.
10     * by John Byrne - R00050076
11     */
12     public class TweetCellRenderer implements ListCellRenderer {
13
14         @Override
15         public Component getListCellRendererComponent(JList list, Object value, int index,
16                                                         boolean isSelected, boolean cellHasFocus) {
17
18             //This is the Colour used when a cellHasFocus
19             Color cellSelectedColor = new Color (190, 193, 199);

```

Fig 33. An example of a commented class.

To the very best of my knowledge. I have adhered to programming standards in so far as variable naming conventions etc. and have created 6 classes in order to tackle this project in a structured way, as outlined in the Class Diagram at the beginning of this document.

4.10 REQ 3(B): THIS DOCUMENT STATING WHAT DONE INCLUDING SCREEN SHOTS OF CODE SNIPPETS & THE RUNNING APP CODE (ON PAGES 3+). APP SPECIFICATION (~½ PAGE). LIST OF CLASSES AND WHAT EACH IS FOR. EVALUATION (~½ PAGE IN 3 BULLET POINTS): HOW HAS IT WORKED OUT, WHAT COULD BE DONE DIFFERENTLY & FUTURE OUTLOOK.

Each requirement has been addresses above with Screen shots and code snippets as required. The full source code is included in the Appendices. App specification is part of the cover page and the List of classes is done as a Class diagram at the beginning of this document. The Evaluation is the next Section.

5 EVALUATION AND CONCLUSION

- How it worked out:

Thankfully, this project has come together nicely. At first, I knew absolutely nothing about Twitter and probably still know very little about twitter, but the complication here was that I had no knowledge for what a Twitter client might look like. After seeing a few prototypes from other students and gaining a better understand of the requirements from Karl, I was able to envision a concept that would satisfy the requirements. The GUI Form designer, as supplied with the IntelliJ IDE, was a great tool to 'trial and error' a number of GUI layouts in terms of Look and Feel. The ListCellRenderer was a complicated piece of functionality which I had no previous experience with, but gaining an understanding of this has allowed me to integrate profile pictures and Favourite / Retweeted icons in my tweet feed, which has added to the overall professionalism of the program.

- What could be done differently:

Not a huge amount of effort was put into following any particular programming model, such as model-view-controller; and while it would not be too difficult to adapt the current program to follow this format, the deadline is fast approaching and this is not a requirement. I would have also like the possibility of integrating my twitter client, with twitter via the Java API, but again this is another learning curve which I simply didn't have time for this semester.

- Future outlook:

If I was to look at this again in the future, then the Twitter API integration as mentioned previous, would be a no brainer. Additional functionality which could be considered would be...

- Added ability to handle multiple users.
- Perhaps even the ability to manually select various Look and Feel themes as an option through the Menu system.
- Ability to load and save session to different filenames
- Ability to toggle multiple tweets' favourite flag.

6 APPENDIX A - COMPLETE SOURCE CODE

6.1 MAIN.JAVA

```
import LoginGui.LoginForm;

import javax.swing.*;

/**
 * Application User Interfaces Assignment II
 * by John Byrne - R00050076
 *
 * Twitter Client Mock-up Application Launcher
 */
public class Main {
    public static void main(String[] args){

        //Setup my Look and Feel theme
        setTheme();

        //LoginForm loginForm = new LoginForm();
        new LoginForm();

    }

    private static void setTheme(){
        try {
            for (UIManager.LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (Exception e) {
            // If Nimbus is not available, you can set the GUI to another look and feel.
        }
    }
}
```

6.2 LOGINFORM.JAVA

```

package LoginGui;

import TwitterClientGui.TwitterClient;
import org.apache.commons.lang3.text.WordUtils;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * Mock Twitter Client
 * by John Byrne - R00050076
 * Login Page
 */
public class LoginForm extends JFrame{
    private JPanel loginPanel;
    private JButton btnLogin;
    private JTextField txtUsername;
    private JPasswordField txtPassword;
    private int loginAttempts; //Track the number of log-ins attempted in this
    session.

    public LoginForm(){
        super("App UI Twitter Mock-up - Login Screen");
        setContentPane(loginPanel);
        pack();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Initialise instance variable;
        loginAttempts = 0;

        /**
         * Only one action is performed on this window; to validate the users' login attempt.
         */
        btnLogin.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                loginAttempts++;
                String username = txtUsername.getText();
                String password = String.valueOf(txtPassword.getPassword());

                if (username.equals("test") && password.equals("test")) {
                    setVisible(false);
                    JOptionPane.showMessageDialog(LoginForm.this, "Welcome " +
                        WordUtils.capitalize(username) + "!");
                    new TwitterClient();
                } else {
                    if (loginAttempts < 3) {
                        JOptionPane.showMessageDialog(LoginForm.this,
                            "Access Denied!\nAttempts remaining: " + (3 - loginAttempts));
                    } else { //On third failed login attempt.
                        JOptionPane.showMessageDialog(LoginForm.this,
                            "Access Denied!\nToo many failed Login attempts!\nGoodbye!");
                        System.exit(0);
                    }
                }
            }
        });

        setSize(360, 160);
        setResizable(false);
        setLocationRelativeTo(null);
        setVisible(true);
    }
}

```


6.3 TWITTERCLIENT.JAVA

```

package TwitterClientGui;

import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.ArrayList;
import java.util.Random;

/**
 * Application User Interfaces Assignment II
 * by John Byrne - R00050076.
 */
public class TwitterClient extends JFrame {
    private JPanel twitterClient;
    private JButton btnSendTweet;
    private JList lstTweets;
    private JTextArea txtTweet;
    private JButton btnFavourite;
    private JButton btnRetweet;
    private JLabel lblOf140Chars;

    private ArrayList<Tweet> tweetFeedFromFile; //Transition ArrayList. Store the tweets loaded
    // from file,
    // before they are passed into the
    defaultListModel
    DefaultListModel defaultListModel; //The data model for the JList
    ArrayList<Tweet> builtInTweets; //An ArrayList storing 'built-in tweets' to
    // simulate a feed.
    Timer timer; //Used to spread-out when the 'built-in tweets'
    // are added.

    public TwitterClient() {
        super("App UI Twitter Mock-up - Client");

        //// Initialise may variables
        tweetFeedFromFile = new ArrayList<>();
        defaultListModel = new DefaultListModel();
        builtInTweets = new ArrayList<>();

        //// Configure the JFrame
        setContentPane(twitterClient);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //// JList Formatting...How the Live Feed looks and feels...
        lstTweets.setCellRenderer(new TweetCellRenderer());
        lstTweets.setModel(defaultListModel);

        ////Read in the auto Tweets from file.
        loadAutoTweetsFromFile();

        ////Read in the initial Tweets from file.
        loadSessionFromFile("initialTweetSession.twc");

        //// Setup Menu and MenuItems
        menuSetup();

        //// Setup Button Action Listeners
        buttonSetup();

        //// Setup the new Tweet TextArea
        setupTweetTextArea();

        //// Setup the autoTweet timer
        setupAutoTweetTimer();

        /*
        //// TESTING
        defaultListModel.add(0, new Tweet("@John Byrne", "This is a test message", false,
        false));
        defaultListModel.add(0, new Tweet("@Bob_Hope", "Something that Bob would say...", true,
        true));
        */
    }

```

```

        defaultListModel.add(0, new Tweet("@Ringo_Starr", "I say this with peace and love,
peace and love!", true, false));
        defaultListModel.add(0, new Tweet("@Bob_Hope", "<html>Whether it is Twitter, Facebook,
Yelp or just <br>a post to co-workers or business officials, the <br>number of actual
characters matters. Bob Build</html>", true, false));
        defaultListModel.add(0, new Tweet("@Jimi_Hendrix ", "<html>Whether it is Twitter,
Facebook, Yelp or just<br> a post to co - workers or business officials, the <br>number of
actual characters matters. Bob Build</html>", true, false));
    */

    /// Finish JFrame prep...
    pack();
    setSize(560, 600);
    setResizable(false);
    setLocationRelativeTo(null);
    setVisible(true);
}

/**
 * The method adds a DocumentListener to the TextArea.
 * The Document listener acts upon changes to the textArea and updates the character
counter.
 */
private void setupTweetTextArea() {
    //////////////////////////////////////
    /// Tweet textarea configuration ///
    //////////////////////////////////////
    txtTweet.getDocument().addDocumentListener(new DocumentListener() {
        @Override
        public void insertUpdate(DocumentEvent e) {
            lblOf140Chars.setText(String.valueOf("Characters: " +
e.getDocument().getLength()) + "/140");
            if (e.getDocument().getLength() > 140) {
                lblOf140Chars.setForeground(Color.RED);
            }
            //Characters: 0 out of a maximum 140
        }

        @Override
        public void removeUpdate(DocumentEvent e) {
            lblOf140Chars.setText(String.valueOf("Characters: " +
e.getDocument().getLength()) + "/140");
            if (e.getDocument().getLength() <= 140) {
                lblOf140Chars.setForeground(Color.BLACK);
            }
        }

        @Override
        public void changedUpdate(DocumentEvent e) {
        }
    });
}

/**
 * This method sets up the Buttons and their ActionListeners
 */
private void buttonSetup() {

    /**
     * Favourite Button Action Listener
     * Sets the isFav flag of the selected Tweet(s)
     * The TweetCellRenderer looks after updating the Favourite icon once repaint() is
called.
     */
    btnFavourite.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            int[] selectedItems = lstTweets.getSelectedIndices();
            for (int item : selectedItems){
                Tweet tweet = (Tweet) defaultListModel.get(item);
                if (tweet.isFav()){
                    tweet.setIsFav(false);
                } else {

```

```

        tweet.setIsFav(true);
    }
    repaint();
}

}

});

/**
 * Re-Tweet Button Action Listener
 * Sets the isReTweeted flag of the selected Tweet.
 * The TweetCellRenderer looks after updating the Re-Tweeted
 * icon once the re-tweeted tweet is displayed.
 */
btnReTweet.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        int[] selectedItems = lstTweets.getSelectedIndices();
        if (selectedItems.length > 1) {
            JOptionPane.showMessageDialog(TwitterClient.this,
                "Please select only one Tweet to re-Tweet. Tweet, Tweet! :)");
        } else if (selectedItems.length > 0) {
            for (int item : selectedItems) {
                Tweet tweet = (Tweet) defaultListModel.get(item);
                if (!tweet.isReTweeted()) {
                    tweet.setIsReTweeted(true);
                }
                defaultListModel.add(0, new Tweet("@John_Byrne", "<html>" +
                    tweet.getFromUser() +
                        " said: <br>" + tweet.getTweetMessage(), false, false));
            }
        }
    }
});

/**
 * Send Tweet Button Action Listener
 * If the message is not too long, it takes the content
 * of the Tweet TextArea and adds a new Tweet to the feed.
 */
btnSendTweet.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (txtTweet.getDocument().getLength() > 140) {
            JOptionPane.showMessageDialog(TwitterClient.this,
                "Your tweet is too long!\n Maximum of 140 characters.");
        } else if (txtTweet.getDocument().getLength() > 0) {
            String formattedTweet = formatTweet(txtTweet.getText());
            defaultListModel.add(0, new Tweet("@John_Byrne", formattedTweet, false,
false));

            txtTweet.setText(""); //Blank out the new tweet text box, ready for a new
entry.

        }
    }
});

/**
 * This method sets up the Menu, MenuItems and their ActionListeners
 */
private void menuSetup() {

    ///////////////////////////////////
    /// Menu Items ///
    ///////////////////////////////////
    JMenuBar menuBar = new JMenuBar();
    setJMenuBar(menuBar);

    JMenu file = new JMenu("File");
    menuBar.add(file);
    JMenuItem open = new JMenuItem("Open");

```

```

file.add(open);
JMenuItem save = new JMenuItem("Save");
file.add(save);
file.addSeparator();
JMenuItem exit = new JMenuItem("Exit");
file.add(exit);

JMenu help = new JMenu("Help");
menuBar.add(help);
JMenuItem about = new JMenuItem("About");
help.add(about);

////////////////////////////////////
//// Menu Item Action Listeners ////
////////////////////////////////////
/**
 * Exit MenuItem Action Listener
 * Exits the application
 */
exit.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});

/**
 * About MenuItem Action Listener
 * Shows an 'About' Dialogue box.
 */
about.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(TwitterClient.this, "Application User Interfaces
Assignment 2\n" +
            "by John Byrne - R00050076\n" +
            "Twitter Client Mock-up v1");
    }
});

/**
 * Open MenuItem Action Listener.
 * It produces an Option Dialogue box to get confirmation.
 * If confirmed, it loads the saved twitter feed from file.
 */
open.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        //Create an OK / Cancel Dialogue box
        Object[] options = {"OK", "Cancel"};
        int overwriteConfirmation = JOptionPane.showOptionDialog(TwitterClient.this,
            "** Warning ** Any unsaved tweets will be lost. Continue?", "Confirm
overwrite session ",
            JOptionPane.PLAIN_MESSAGE,
            JOptionPane.QUESTION_MESSAGE,
            null,
            options,
            options[0]);

        if (overwriteConfirmation == 0) {
            loadSessionFromFile("savedTweetSession.twc");
        }
    }
});

/**
 * Save MenuItem Action Listener.
 * It produces an Option Dialogue box to get confirmation.
 * If confirmed, it saves the twitter feed to the "savedTweetSession.twc" file.
 */
save.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

```

```

//Create an OK / Cancel Dialogue box
Object[] options = {"OK", "Cancel"};
int overwriteConfirmation = JOptionPane.showOptionDialog(TwitterClient.this,
    "** Warning ** Previously saved session will be overwritten.
Continue?",
    "Confirm overwrite session ",
    JOptionPane.PLAIN_MESSAGE,
    JOptionPane.QUESTION_MESSAGE,
    null,
    options,
    options[0]);

if (overwriteConfirmation == 0) {
    ArrayList<Tweet> sessionTweets = new ArrayList<>();

    for (int index = 0; index < defaultListModel.getSize(); index++) {
        Tweet tweet = (Tweet) defaultListModel.get(index);
        sessionTweets.add(tweet);
    }

    try {
        FileOutputStream fos = new FileOutputStream("savedTweetSession.twc");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(sessionTweets);
        oos.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}

});
}

/**
 * This method loads the Auto-Tweets in the to Twitter Client GUI.
 * This method is only called at start-up
 */
private void loadAutoTweetsFromFile() {
    FileInputStream fis = null;
    try {
        fis = new FileInputStream("autoTweets.twc");
        ObjectInputStream ois = new ObjectInputStream(fis);
        builtInTweets = (ArrayList<Tweet>) ois.readObject();
        ois.close();
    } catch (IOException ex) {
        ex.printStackTrace();
        //Could throw in a Dialogue box to say there was an issue, but I don't think it's
        necessary for this app.
    } catch (ClassNotFoundException ex) {
        ex.printStackTrace();
    }
}

/**
 * Starts a background timer that will add a random Auto-Tweet at a random interval,
 * between 10 and 25 seconds.
 */
private void setupAutoTweetTimer() {
    Random random = new Random();
    timer = new Timer(random.nextInt(15000) + 10000,
        new ActionListener() { //Every 10 - 25 seconds, do this...
            @Override
            public void actionPerformed(ActionEvent e) {

                defaultListModel.add(0,
                    builtInTweets.get(random.nextInt(builtInTweets.size())));

            }
        });
    timer.start();
}

/**
 * This method loads the content of the .twc file and adds the contained tweets to the
 * Client view.
 * @param filename The filename to load

```

```

*/
private void loadSessionFromFile(String filename) {
    FileInputStream fis = null;
    try {
        fis = new FileInputStream(filename);
        ObjectInputStream ois = new ObjectInputStream(fis);
        tweetFeedFromFile = (ArrayList<Tweet>) ois.readObject();
        ois.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    } catch (ClassNotFoundException ex) {
        ex.printStackTrace();
    }

    defaultListModel.removeAllElements();
    tweetFeedFromFile.forEach(defaultListModel::addElement);
}

/**
 * This method takes in a String and adds html <br> tags to it,
 * to allow for multiple lines in the Tweet Cell JLabel
 * @param tweetText The source Tweet text
 * @return The returned multi-lined html text
 */
private String formatTweet(String tweetText) {
    // Need to format the tweet, so that it fits properly in the feed.
    // I try to split the line at the end of a whole word, where possible.
    int tweetLength = tweetText.length();
    if (tweetLength > 140) {
        if (tweetText.indexOf(' ', 140) < 155 && tweetText.indexOf(' ') != -1) {
            tweetText = new StringBuilder(tweetText).insert(tweetText.indexOf(' ', 140),
"<br>").toString();
        } else {
            tweetText = new StringBuilder(tweetText).insert(140, "<br>").toString();
        }
    }
    if (tweetLength > 94) {
        if (tweetText.indexOf(' ', 94) < 105 && tweetText.indexOf(' ') != -1) {
            tweetText = new StringBuilder(tweetText).insert(tweetText.indexOf(' ', 94),
"<br>").toString();
        } else {
            tweetText = new StringBuilder(tweetText).insert(94, "<br>").toString();
        }
    }
    if (tweetLength > 47) {
        if (tweetText.indexOf(' ', 47) < 58 && tweetText.indexOf(' ') != -1) {
            tweetText = new StringBuilder(tweetText).insert(tweetText.indexOf(' ', 47),
"<br>").toString();
        } else {
            tweetText = new StringBuilder(tweetText).insert(47, "<br>").toString();
        }
    }
    return "<html>" + tweetText + "</html>";
}

/**
 * The method is not used in normal runtime.
 * It is used to create the autoTweets.twc file which is read in when the program starts
 * to populate the builtInTweets ArrayList.
 */
private void makeAutoTweets() {
    ArrayList<Tweet> builtInTweets = new ArrayList<>();
    builtInTweets.add(new Tweet("@Sean_Connery", "<html>...Bond. James Bond.</html>",
false, false));
    builtInTweets.add(new Tweet("@Humphrey_Bogart", "<html>Of all the gin joints in all the
towns in all the<br> world, she walks into mine.</html>", false, false));
    builtInTweets.add(new Tweet("@Arnold_Schwarz", "<html>I'll be back.</html>", false,
false));
    builtInTweets.add(new Tweet("@Tom_Hanks", "<html>My Mama always said, 'Life was like a
box of<br> chocolates; you never know what you're gonna get.</html>", false, false));
    builtInTweets.add(new Tweet("@Robert_De_Niro", "<html>You talkin' to me? Well, who the
hell else are<br> you talkin' to? Well, I'm the only one here.<br> Who the f--k do you think
you're talkin' to?</html>", false, false));
    builtInTweets.add(new Tweet("@Arnold_Schwarz", "<html>Hasta la vista, baby.</html>",
false, false));
    builtInTweets.add(new Tweet("@Bruce_Willis", "<html>Yippie-Kai-Yay,

```

```

Motherfucker!</html>", false, false));
    builtInTweets.add(new Tweet("@Ringo_Starr", "<html>I say this with peace and love,
peace and love!</html>", false, false));
    builtInTweets.add(new Tweet("@Bob_Hope", "<html>Whether it is Twitter, Facebook, Yelp
or just <br>a post to co-workers or business officials, the <br>number of actual characters
matters. Bob Build</html>", false, false));
    builtInTweets.add(new Tweet("@Jimi_Hendrix", "<html>Whether it is Twitter, Facebook,
Yelp or just<br> a post to co - workers or business officials, the <br>number of actual
characters matters. Bob Build</html>", false, false));

    try {
        FileOutputStream fos = new FileOutputStream("autoTweets.twc");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(builtInTweets);
        oos.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
}

```

6.4 TWEETCELLRENDERER.JAVA

```

package TwitterClientGui;

import javax.swing.*.*;
import java.awt.*.*;

/**
 * My Tweet Cell Renderer class
 * Each cell in my JList is rendered based on the rules defined here.
 * by John Byrne - R00050076
 */
public class TweetCellRenderer implements ListCellRenderer {

    @Override
    public Component getListCellRendererComponent(JList list, Object value, int index,
                                                  boolean isSelected, boolean cellHasFocus) {

        //This is the Colour used when a cellHasFocus
        Color cellSelectedColor = new Color (190, 193, 199);

        //Get the data and cast it as a Tweet object
        Tweet tweet = (Tweet)value;
        /* From Tweet object
         this.fromUser = fromUser;
         this.tweetMessage = tweetMessage;
         this.isFav = isFav;
         this.isReTweeted = isReTweeted;
         */

        //Extract all the data from the Tweet...
        String fromUser = tweet.getFromUser();
        String tweetMessage = tweet.getTweetMessage();
        boolean isFav = tweet.isFav();
        boolean isReTweeted = tweet.isReTweeted();

        //Build the Tweet cell...
        TweetCellTemplate cell = new TweetCellTemplate(); //Create an instance of my Cell
        template, which I will                                //ultimately pass back to the JList
        as an element.

        ImageIcon icon = createImageIcon("../Images/" + fromUser + ".png", "User Profile
        Picture");

        cell.getLblProfilePic().setIcon(icon); //Get profile picture
        cell.getLblProfilePic().setText(fromUser); //Set the profile picture
        cell.getLblTweet().setText(tweetMessage); //Set the profile name
        cell.getLblTweet().setText(tweetMessage); //Set the Tweet message

        //The cells default to isFav and isReTweeted == false, so if I set isFav to true and
        back again to false,
        //the cell will return to default configuration, so I don't need to repaint the
        isNotFav icon at runtime.
        if(isFav){
            ImageIcon favIcon = createImageIcon("../Images/isFavStar.png", "a Favourite
            Tweet");
            cell.getLblFav().setIcon(favIcon);
        }
        if(isReTweeted){
            ImageIcon reTweetIcon = createImageIcon("../Images/isReTweeted.png", "a ReTweeted
            Tweet");
            cell.getLblReTweet().setIcon(reTweetIcon);
        }

        //if(isSelected){ //Commented this out as I found callHasFocus works better for this
        application.
        cell.setBackground(Color.GREEN);
        }*/

        if(cellHasFocus){
            //Color origCol = cell.getBackground();
            //cell.getLblReTweet().setText(origCol.toString());
            cell.setBackground(cellSelectedColor);
        }
    }
}

```



```

        return cell;
    }

    /**
     * This method returns the ImageIcon to be used as the profile picture in my Tweet cell.
     * @param path The Path of the profile pic
     * @param description The description associated with the profile pic.
     * @return ImageIcon for use as Tweet profile pic.
     */
    protected ImageIcon createImageIcon(String path, String description) {

        String defaultPath = "../Images/noProfilePictureConfigured.png"; //To allow for
        profiles with no Pic

        java.net.URL imgURL = getClass().getResource(path);
        if (imgURL != null) {
            return new ImageIcon(imgURL, description);
        } else { //If I can't find a profile pic, use the default one...
            imgURL = getClass().getResource(defaultPath);
            return new ImageIcon(imgURL, description);
        }
    }
}

```

6.5 TWEETCELLTEMPLATE.JAVA

```

package TwitterClientGui;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;

/**
 * This TweetCellTemplate class is a customised JPanel for use in my JList.
 * by John Byrne - R00050076
 */
public class TweetCellTemplate extends JPanel {
    private JPanel rootTweet;
    private JLabel lblProfilePic;
    private JLabel lblTweet;
    private JLabel lblFav;
    private JLabel lblReTweet;

    /**
     * This constructor builds the JList Element.
     */
    public TweetCellTemplate() {
        setBorder(new EmptyBorder(5, 5, 5, 5));
        setLayout(new GridBagLayout());
        GridBagConstraints gridBagConstraints = new GridBagConstraints();

        gridBagConstraints.anchor = GridBagConstraints.WEST;
        gridBagConstraints.weightx = 1;
        add(lblProfilePic, gridBagConstraints);

        add(lblTweet); //added without gridBagConstraints so that it will float

        gridBagConstraints.anchor = GridBagConstraints.EAST;
        add(lblFav, gridBagConstraints);
        add(lblReTweet);
    }

    public void setColor(Color color) {
        this.setBackground(color);
    }

    public JLabel getLblProfilePic() {
        return lblProfilePic;
    }

    public JLabel getLblTweet() {
        return lblTweet;
    }

    public JLabel getLblFav() {
        return lblFav;
    }

    public JLabel getLblReTweet() {
        return lblReTweet;
    }
}

```

6.6 TWEET.JAVA

```

package TwitterClientGui;

import java.io.Serializable;

/**
 * Created by jbyrne on 12/05/2015.
 */
public class Tweet implements Serializable {
    private String fromUser;        //Who is sending the tweet @John_Byrne
    private String tweetMessage;    //The tweet being sent
    private boolean isFav;          //Has the tweet been marked as a Favourite?
    private boolean isRetweeted;    //Has the tweet been marked as Retweeted?

    public Tweet(String fromUser, String tweetMessage, boolean isFav, boolean isRetweeted) {
        this.fromUser = fromUser;
        this.tweetMessage = tweetMessage;
        this.isFav = isFav;
        this.isRetweeted = isRetweeted;
    }

    /// Getters and Setters
    public String getFromUser() {
        return fromUser;
    }

    public void setFromUser(String fromUser) {
        this.fromUser = fromUser;
    }

    public String getTweetMessage() {
        return tweetMessage;
    }

    public void setTweetMessage(String tweetMessage) {
        this.tweetMessage = tweetMessage;
    }

    public boolean isFav() {
        return isFav;
    }

    public void setIsFav(boolean isFav) {
        this.isFav = isFav;
    }

    public boolean isRetweeted() {
        return isRetweeted;
    }

    public void setIsRetweeted(boolean isRetweeted) {
        this.isRetweeted = isRetweeted;
    }
}

```