Object oriented Programming (ECM1410) CA3
Printout

March 2021

1 Classes

1.1 Account

```java
package socialmedia;

import java.io.Serializable;
import java.util.HashSet;
import java.util.Set;

/**
 * This class contains all the methods and attributes of the Account and therefore
 * is responsible for handling all the details related to a user on the Social Media
 * Platform.
 * <p>
 * An Account has a unique ID (sequencially incrementting acording to num
 * of users on the system), a string handle (username) and a description similar to
 * a bio.
 * <p>
 * The class itself has a static value for its SerialVersionUID used when
 * serializing the platform, a num of accounts which tracks the number of accounts
 * created and a HashSet of all usernames (handles).
 *
 * @author 700008432
 * @author 690033172
 * @version 1.0
 */
public class Account implements Serializable {

    /**
     * The SerialVersionUID which represents the class version.
     */
    private static final long serialVersionUID = 8225229775036968396L;

```

```
31      /**
32       * This is the unique sequential identifier of the account.
33       */
34      private int id;
35
36      /**
37       * This is the handle attributed to the account.
38       */
39      private String handle;
40
41      /**
42       * This is the description that the user can add to their account.
43       */
44      private String description;
45
46      /**
47       * This is the number of posts, including endorsements and replies the
48       * account has.
49       */
50      private int numPosts = 0;
51
52      /**
53       * This is the number of endorsements the account has recieved.
54       */
55      private int numEndorsements = 0;
56
57      /**
58       * This is the static set of usernames in the system, it contains
59       * all the handles of each user within the system.
60       */
61      private static final Set<String> usernames = new HashSet<String>();
62
63      /**
64       * This is a static integer value which records the number of accounts
65       * in the system starting at 0.
66       */
67      private static int numAccounts = 0;
68
69      /**
70       * This is the constructor method for an Account, it takes in the parameter handle which
71       * will be the username of the account. This constructor is for when the user does not
72       * want to add a description on the creation of their account.
73       *
74       * @param handle    The username the user wants to use.
75       *
76       * @throws IllegalHandleException   Thrown if the handle is already in use.
77       * @throws InvalidHandleException   Thrown if the handle is over 30 characters,
78       *                                  is empty, or contains white space.
```

```
79          */
80         public Account(String handle)
81                             throws IllegalHandleException,
82                             InvalidHandleException{
83             if (!usernames.add(handle)) {
84                 throw new IllegalHandleException("Handle is already in use in the system.");
85             }
86             if (handle.length() > 30) {
87                 throw new InvalidHandleException("Your handle is longer than 30 characters.");
88             }
89             if (handle.contains(" ")) {
90                 throw new InvalidHandleException("The handle cannot contain white space.");
91             }
92             if (handle.equals("")) {
93                 throw new InvalidHandleException("The handle cannot be empty.");
94             }
95             this.handle = handle;
96             this.id = numAccounts++;
97         }
98 
99         /**
100          * This is the constructor method for an Account, it takes in the parameters handle which
101          * will be the username of the account, and the description which will be connected to the
102          * account.
103          *
104          * @param handle        The username the user wants to use.
105          * @param description   The description the user wants to display.
106          *
107          * @throws IllegalHandleException   Thrown if the handle is already in use.
108          * @throws InvalidHandleException   Thrown if the handle is over 30 characters,
109          *                                  is empty, or contains white space.
110          */
111         public Account(String handle, String description)
112                                 throws IllegalHandleException,
113                                 InvalidHandleException {
114             if (!usernames.add(handle)) {
115                 throw new IllegalHandleException("Handle is already in use in the system. ");
116             }
117             if (handle.length() > 30) {
118                 throw new InvalidHandleException("Your handle is longer than 30 characters.");
119             }
120             if (handle.contains(" ")) {
121                 throw new InvalidHandleException("The handle cannot contain white space.");
122             }
123             if (handle.equals("")) {
124                 throw new InvalidHandleException("The handle cannot be empty.");
125             }
126             this.handle = handle;
```

```
127             this.description = description;
128             this.id = numAccounts++;
129         }
130 
131         /**
132          * This method resets the number of posts to 0.
133          */
134         public static void resetNumAccounts() {
135             numAccounts = 0;
136         }
137 
138         /**
139          * This method is for allowing the user the change their handle after creating their
140          * account. It does all the checks to be sure there are no other users with the new handle.
141          *
142          * @param newHandle     This is the handle that the user wants to change to.
143          *
144          * @throws InvalidHandleException   This exception is thrown if the handle is under 30 characters,
145          *                                  is blank, or contains white space.
146          */
147         public void changeHandle(String newHandle)
148                                 throws InvalidHandleException {
149             if (newHandle.length() > 30) {
150           throw new InvalidHandleException("Your handle is longer than 30 characters.");
151         }
152         if (newHandle.contains(" ")) {
153           throw new InvalidHandleException("The handle cannot contain white space.");
154         }
155         if (newHandle.equals("")) {
156           throw new InvalidHandleException("The handle cannot be empty.");
157         }
158 
159             this.handle = newHandle;
160         }
161 
162         /**
163          * This method allows the user to update their account description.
164          *
165          * @param description The new description that they want to add.
166          */
167         public void updateDescription(String description) {
168             this.description = description;
169         }
170 
171         /**
172          * This method adds one to the number of posts, including endorsements and replies the
173          * account has.
174          */
```

```
175     public void addNumPost() {
176         numPosts++;
177     }
178 
179     /**
180      * This method adds one to the number of endorsements the account has.
181      */
182     public void addNumEndorse() {
183         numEndorsements++;
184     }
185 
186     /**
187      * This method minuses one to the number of posts, including endorsements and replies the
188      * account has.
189      */
190     public void minusNumPost() {
191         numPosts--;
192     }
193 
194     /**
195      * This method minuses one to the number of endorsements the account has.
196      */
197     public void minusNumEndorse() {
198         numEndorsements--;
199     }
200 
201     /**
202      * This method returns the handle of the user.
203      *
204      * @return The handle of the user.
205      */
206     public String getHandle() {
207         return handle;
208     }
209 
210     /**
211      * This method returns the description connected to the account.
212      *
213      * @return The description of the account.
214      */
215     public String getDescription() {
216         return description;
217     }
218 
219     /**
220      * This method returns the id of the user.
221      *
222      * @return The Id of the account is what gets returned.
```

```
223         */
224         public int getId() {
225             return id;
226         }
227 
228         /**
229          * This method returns the number of posts, including endorsements and replies an account has.
230          *
231          * @return the number of posts, including endorsements and replies an account has.
232          */
233         public int getNumPosts() {
234             return numPosts;
235         }
236 
237         /**
238          * This method returns the number of endorsements an account has.
239          *
240          * @return the number of endorsements an account has.
241          */
242         public int getNumEndorsements() {
243             return numEndorsements;
244         }
245 
246         /**
247          * This method allows Accounts to be printed out as text that includes
248          * all useful information.
249          *
250          * @return The string of text is returned.
251          */
252         public String toString() {
253             String descToShow = "";
254             if (description != null) {
255                 descToShow += description;
256             }
257             return "ID: " + id + "\nHandle: " + handle + "\nDescription: " +
258                 descToShow+"\nPost count: "+numPosts+"\nEndorse count: "+numEndorsements;
259         }
260     }
```

1.2 Post

```java
package socialmedia;

import java.io.Serializable;

/**
 * This class consists of the methods and attributes of a Post and therefore is
 * responsible for all the details regarding a Post and due to how a Comment and
 * Endorsement inherit from Post it is respinsible for some methods ad details of
 * a a Comment and Endorsement.
 * <p>
 * A Post had a unique sequencial ID, a message up to 100 characters long, an
 * author (Account) it is associated with, a list of all comments it recieves
 * and a list of all endorsements it recieves.
 * <p>
 * The class itself has two static values which are the serialVersionUID which is
 * used when serializing the platform and numPosts which tracks the number of posts
 * which have been created.
 *
 * @author 700008432
 * @author 690033172
 * @version 1.0
 */
public class Post implements Serializable {

    /**
     * The SerialVersionUID which represents the class version.
     */
    private static final long serialVersionUID = -6471649759148628316L;

    /**
     * This is the unique sequencial identifier of the post.
     */
    private int id;

    /**
     * This is the message with up to 100 characters.
     */
    private String message;

    /**
     * This is the Account associated with the post.
     */
    private Account author;

    /**
```

```
46          * This is an array of all the comments the post receieves.
47          */
48         private Comment[] allComments = {};
49
50         /**
51          * This is an array of all the endorsements the post receieves.
52          */
53         private Endorsement[] allEndorsements = {};
54
55
56         private boolean isActionable;
57
58         /**
59          * This is a staic integer value which records the number of posts in the system starting
60          * at 0, it is used to get the next sequencial ID for when a post is created.
61          */
62         private static int numPosts = 0;
63
64         /**
65          * This is the constructor method for a Post, it takes in the parameters author which refers
66          * to the account associated with the post and message which must be up to 100 characters long
67          * for a new post or comment.
68          *
69          * @param author    The account associated with the post.
70          * @param message   The message contained in the post.
71          *
72          * @throws InvalidPostException             Thrown if the message is longer than 100 characters.
73          * @throws HandleNotRecognisedException     Thrown if an Account with given handle does not exist.
74          */
75         public Post(Account author, String message)
76                     throws InvalidPostException,
77                     HandleNotRecognisedException {
78             if (message.length() > 100) {
79                 throw new InvalidPostException("Post has a length of more than 100 characters.");
80             }
81             if (message.equals("")) {
82                 throw new InvalidPostException("Post cannot be empty.");
83             }
84             if (author == null) {
85                 throw new HandleNotRecognisedException("Account with given handle does not exist.");
86             }
87             this.author = author;
88             this.message = message;
89             this.id = numPosts++;
90             this.isActionable = true;
91         }
92
93         /**
```

```
94          * This is the constructor method for a Post, it takes in the parameters author which refers
95          * to the account associated with the post and linkedPost which is the post the new post is associated
96          * with when creating an endorsement post.
97          *
98          * @param author        The account associated with the post.
99          * @param linkedPost    The post associated with this endorsement post.
100         *
101         * @throws HandleNotRecognisedException     Thrown if an Account with given handle does not exist.
102         */
103        public Post(Account author, Post linkedPost)
104                    throws HandleNotRecognisedException {
105            if (author == null) {
106                throw new HandleNotRecognisedException("Account with given handle does not exist.");
107            }
108            this.author = author;
109            String linkedAuthorName = linkedPost.getAuthor().getHandle();
110            String linkedPostMessage = linkedPost.getMessage();
111            if ((linkedPostMessage.length() + linkedAuthorName.length() + 5) > 100) {
112                linkedPostMessage = linkedPostMessage.substring(0, 95-linkedAuthorName.length());
113            }
114            this.message = "EP@"+linkedAuthorName+": "+linkedPostMessage;
115            this.id = numPosts++;
116            this.isActionable = false;
117        }
118
119        /**
120         * This methods adds a new comment to the post's list of comments it has recieved.
121         *
122         * @param newComment    The new comment to add to its list.
123         */
124        public void addComment(Comment newComment) {
125            Comment[] newList = new Comment[allComments.length + 1];
126            for (int i = 0; i < allComments.length; i++) {
127                newList[i] = allComments[i];
128            }
129            newList[allComments.length] = newComment;
130            allComments = newList;
131        }
132
133        /**
134         * This method adds a new endorsement to the post's list of endorsements it has recieved.
135         *
136         * @param newEndorsement    The new endorsement to add to its list.
137         */
138        public void addEndorsement(Endorsement newEndorsement) {
139            Endorsement[] newList = new Endorsement[allEndorsements.length + 1];
140            for (int i = 0; i < allEndorsements.length; i++) {
141                newList[i] = allEndorsements[i];
```

```
142             }
143             newList[allEndorsements.length] = newEndorsement;
144             allEndorsements = newList;
145         }
146 
147         /**
148          * This method emptys a post. This includes removing the author a post refers to, replacing
149          * the messsage with the string "The original content was removed from the system and is no longer
150          * available." and making the post non-actioanble.
151          */
152         public void empty() {
153             this.message = "The original content was removed from the system and is no longer available.";
154             this.author = null;
155             this.isActionable = false;
156         }
157 
158         /**
159          * This method resets the number of posts to 0.
160          */
161         public static void resetNumPosts() {
162             numPosts = 0;
163         }
164 
165         /**
166          * This method returns the id of the Post.
167          *
168          * @return The id of the Post.
169          */
170         public int getID() {
171             return id;
172         }
173 
174         /**
175          * This method returns the message contained in the Post.
176          *
177          * @return The message contained in the Post.
178          */
179         public String getMessage() {
180             return message;
181         }
182 
183         /**
184          * This method returns the author of the Post.
185          *
186          * @return The author associated with the Post.
187          */
188         public Account getAuthor() {
189             return author;
```

```
190     }
191
192     /**
193      * This method returns all the endorsement posts of the Post.
194      *
195      * @return The array of all endorsements.
196      */
197     public Endorsement[] getAllEndorsements() {
198         return allEndorsements;
199     }
200
201     /**
202      * This method returns all the comments associated with the Post
203      *
204      * @return The array of all comments.
205      */
206     public Comment[] getAllComments() {
207         return allComments;
208     }
209
210     /**
211      * This method returns if the post can be acted upon.
212      *
213      * @return isActionable If the post can be acted upon.
214      */
215     public boolean getIsActionable() {
216         return isActionable;
217     }
218
219     /**
220      * This method returns the post in string format as defined in the MiniSocialMediaPlatform
221      * interface.
222      *
223      * @return The post in string format.
224      */
225     @Override
226     public String toString() {
227         String name = "";
228         if (!(author == null)){
229             name += author.getHandle();
230         }
231         return "ID: "+id+"\nAccount: "+name+"\nNo. endorsements: "+allEndorsements.length+
232             " | No. comments: "+allComments.length+"\n"+message;
233     }
234 }
```

1.3 Comment

```
1  package socialmedia;
2
3  /**
4   * This class consists of the specific methods and attributes of a Comment, some of the other methods and
5   * attributes of a Comment are inheritted from a Post. Therefore this class is responsible for the Comment
6   * specific methods and attributes.
7   * <p>
8   * As a Comment inherits from a Post and thefore has access to all of a Post's attributes
9   * and methods. In addition to these, a Comment has a linkedPost attribute which refers
10  * to the Post which the comment has been placed on.
11  * <p>
12  * The class itself has a static value for its SerialVersionUID used when
13  * serializing the platform.
14  *
15  * @author 700008432
16  * @author 690033172
17  * @version 1.0
18  */
19 public class Comment extends Post{
20
21     /**
22      * The SerialVersionUID which represents the class version.
23      */
24     private static final long serialVersionUID = -7709952939872791336L;
25
26     /**
27      * This is the post the comment is associated with.
28      */
29     private Post linkedPost;
30
31     /**
32      * This is the constructor method for a Comment, it takes in the parameters author which refers
33      * to the account associated with the post, message which must be up to 100 characters long and
34      * a Post which the comment is associated with.
35      *
36      * @param author        The account associated with the comment.
37      * @param message       The message contained in the comment.
38      * @param linkedPost    The Post which the comment has ben added on.
39      *
40      * @throws InvalidPostException             Thrown when the message is longer than 100 characters.
41      * @throws HandleNotRecognisedException     Thrown when the given account ahdnle does not exist in
42      *                                          the system.
43      */
44     public Comment(Account author, String message, Post linkedPost)
45                     throws InvalidPostException,
```

```
46                      HandleNotRecognisedException {
47          super(author, message);
48          this.linkedPost = linkedPost;
49      }
50
51      /**
52       * This method is used to change the post which the comment is associated with.
53       *
54       * @param newLinkedPost     The new post the comment is associated with.
55       */
56      public void changeLinkedPost(Post newLinkedPost) {
57          this.linkedPost = newLinkedPost;
58      }
59
60      /**
61       * This method returns the Post the comment is associated with.
62       *
63       * @return The post the comment is associated with.
64       */
65      public Post getLinkedPost() {
66          return linkedPost;
67      }
68  }
```

1.4 Endorsement

```java
package socialmedia;

/**
 * This class consists of the specific methods and attributes of a Endorsement, some of the
 * other methods and attributes of a Endorsement are inheritted from a Post. Therefore this
 * class is responsible for only the methods and attributes unqiue for an Endorsement.
 * <p>
 * As an Endorsement inherits from a Post and thefore has access to all of a Post's attributes
 * and methods. In addition to these, an Endorsement has a linkedPost attribute which refers
 * to the Post which the endorsement has been placed on and replicates the message of the post
 * it is associated with.
 * <p>
 * The class itself also has a static value for its SerialVersionUID used when
 * serializing the platform.
 *
 * @author 700008432
 * @author 690033172
 * @version 1.0
 */
public class Endorsement extends Post {

    /**
     * The SerialVersionUID which represents the class version.
     */
    private static final long serialVersionUID = 8178673147762741699L;

    /**
     * This is the post the comment is associated with.
     */
    private Post linkedPost;

    /**
     * This is the constructor method for an Endorsement, it takes in the parameters author which refers
     * to the account associated with the post and a Post which the endorsement is associated with.
     * The message of the post it is associated wth is formatted as:
     *
     * <p>
    * <code>"EP@" + [endorsed account handle] + ": " + [endorsed message]</code>
    * <p>
     *
     * @param author        The account associated with the endorsement.
     * @param linkedPost    The Post which the endorsement has ben added on.
     *
     * @throws HandleNotRecognisedException     Thrown if an Account with given handle does not exist.
     */
```

```
46      public Endorsement(Account author, Post linkedPost)
47                          throws HandleNotRecognisedException {
48          super(author, linkedPost);
49          this.linkedPost = linkedPost;
50      }
51
52      /**
53       * This method returns the Post the endorsement is associated with.
54       *
55       * @return The post the endorsement is associated with.
56       */
57      public Post getLinkedPost() {
58          return linkedPost;
59      }
60  }
```

1.5 SocialMedia

```java
package socialmedia;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

/**
 * This class is am implimetation of the SocialMediaPlatform interface which extends the
 * MiniSocialMediaPlatform interface. Therefore it is resposible for most of the functionality
 * of the platform.
 * <p>
 * A SocialMedia object has a list of all accounts and a list of all posts which is used to hold
 * all data on the Social Media Platform. The accounts, posts, endorsments and comments in these
 * lists are then created, removed an edisted using the methods in the class.
 * <p>
 * The class itself also has a static value for its SerialVersionUID used when
 * serializing the platform.
 *
 * @author 700008432
 * @author 690033172
 * @version 1.0
 */
public class SocialMedia implements SocialMediaPlatform {

  /**
   * The SerialVersionUID which represents the class version.
   */
  private static final long serialVersionUID = -5306444784233657143L;

  /**
   * This is an array of all accounts on the platform.
   */
  private Account[] allAccounts = {};

  /**
   * This is an array of all posts on the platform.
   */
  private Post[] allPosts = {};

  /**
   * This method is used to add a instance of an account to the array of all accounts on
   * the platform.
   *
```

```
46      * @param newAccount  The new account to be added to the array.
47      */
48     public void addAccountToAllAccounts(Account newAccount) {
49       Account[] newList = new Account[allAccounts.length + 1];
50       for (int i = 0; i < allAccounts.length; i++) {
51         newList[i] = allAccounts[i];
52       }
53       newList[allAccounts.length] = newAccount;
54       allAccounts = newList;
55     }
56
57     /**
58      * This method is used to delete an account from all the accounts in the system.
59      *
60      * @param deleteAccount  The account to be deleted.
61      */
62     public void deleteAccountFromAllAccounts(Account deleteAccount) {
63       Account[] newList = new Account[allAccounts.length - 1];
64       for (int i = 0, j = 0; i < allAccounts.length; i++) {
65         if (!allAccounts[i].equals(deleteAccount)) {
66           newList[j] = allAccounts[i];
67           j++;
68         }
69       }
70       allAccounts = newList;
71     }
72
73     /**
74      * This method is used to add a instance of a post to the array of all posts on
75      * the platform.
76      *
77      * @param newPost  The new post to be added to the array.
78      */
79     public void addPostToAllPosts(Post newPost) {
80       Post[] newList = new Post[allPosts.length + 1];
81       for (int i = 0; i < allPosts.length; i++) {
82         newList[i] = allPosts[i];
83       }
84       newList[allPosts.length] = newPost;
85       allPosts = newList;
86     }
87
88     /**
89      * This method is used to delete a post from all the posts in the system.
90      *
91      * @param deletePost  The post to be deleted.
92      */
93     public void deletePostFromAllPosts(Post deletePost) {
```

```
94        Post[] newList = new Post[allPosts.length - 1];
95        for (int i = 0, j = 0; i < allPosts.length; i++) {
96          if (!allPosts[i].equals(deletePost)) {
97            newList[j] = allPosts[i];
98            j++;
99          }
100       }
101       allPosts = newList;
102     }
103
104     /**
105      * This method gets a Post from all posts with the given id.
106      *
107      * @param id  The id of the wanted post.
108      *
109      * @return wantedPost The post with the given id.
110      */
111     public Post getPost(int id) {
112       Post wantedPost = null;
113       for (Post post : allPosts) {
114         if (post.getID() == id) {
115           wantedPost = post;
116           break;
117         }
118       }
119       return wantedPost;
120     }
121
122     /**
123      * This method gets an Account from all accounts with the given handle.
124      *
125      * @param handle  The handle of the desired account.
126      *
127      * @return  This returns the account with the matching handle.
128      */
129     public Account getAccount(String handle) {
130       Account wantedAccount = null;
131       for (Account account : allAccounts) {
132         if (account.getHandle().equals(handle)) {
133           wantedAccount = account;
134           break;
135         }
136       }
137       return wantedAccount;
138     }
139
140     /**
141      * This method gets an Account from all accounts with the given id.
```

```
142      *
143      * @param id  The id of the desired account.
144      *
145      * @return  This returns the account with the matching id.
146      */
147     public Account getAccount(int id) {
148       Account wantedAccount = null;
149       for (Account account : allAccounts) {
150         if (account.getId() == id) {
151           wantedAccount = account;
152           break;
153         }
154       }
155       return wantedAccount;
156     }
157
158       /**
159        * This method allows the user to create an account with just their handle
160      * and it returns their id number afterwards. It does the checks on the handle
161      * to ensure that there are no other users with the same handle, the handle is
162      * under 30 characters, is not empty and finally that it has no white space.
163      *
164      * @param handle  This is the handle the user wants to use
165      *
166      * @throws IllegalHandleException  Thrown if the handle is already in use by another user.
167      * @throws InvalidHandleException  Thrown if the handle is over 30 chars, is empty or has white space.
168      *
169      * @return This returns the id of the new account.
170      */
171       @Override
172     public int createAccount(String handle)
173                   throws IllegalHandleException,
174                   InvalidHandleException {
175       Account newAccount = new Account(handle);
176       addAccountToAllAccounts(newAccount);
177       return newAccount.getId();
178     }
179
180       /**
181        * This method allows the user to create an account with the handle and description.
182      * It does checks to ensure the handle is valid and not already in use within the
183      * system.
184      *
185      * @param handle    The handle of the new user.
186      * @param description  The description to be displayed on the profile.
187      *
188      * @throws IllegalHandleException  Thrown if the handle is already in use by another user.
189      * @throws InvalidHandleException  Thrown if the handle is over 30 chars, is empty or has white space.
```

```
190      *
191      * @return This returns the id of the new account.
192        */
193     @Override
194     public int createAccount(String handle, String description)
195                   throws IllegalHandleException,
196                   InvalidHandleException {
197       Account newAccount = new Account(handle, description);
198       addAccountToAllAccounts(newAccount);
199       return newAccount.getId();
200     }
201
202       /**
203        * This method allows the user to delete an account using the account id.
204      *
205      * @param id  The id related to account to be deleted.
206      *
207      * @throws AccountIDNotRecognisedException  Thrown when the account id used is not in the system.
208        */
209     @Override
210     public void removeAccount(int id)
211                   throws AccountIDNotRecognisedException {
212       Account deleteAccount = getAccount(id);
213
214       if (deleteAccount == null) {
215         throw new AccountIDNotRecognisedException("Attemting to delete Account with a given
216             Account ID which does not exist in system.");
217       }
218
219       for (Post post : allPosts) {
220         if (post.getAuthor() == deleteAccount) {
221           deletePostFromAllPosts(post); //deletes each post before removing the account.
222         }
223       }
224       deleteAccountFromAllAccounts(deleteAccount);
225
226     }
227
228       /**
229        * This method allows the user to remove an account using the account handle.
230      *
231      * @param handle  The handle of the account to be removed.
232      *
233      * @throws HandleNotRecognisedException  Throwm when the provided handle is not in the system.
234        */
235     @Override
236     public void removeAccount(String handle)
237                   throws HandleNotRecognisedException {
```

```
238       Account deleteAccount = getAccount(handle);
239
240       if (deleteAccount == null) {
241         throw new HandleNotRecognisedException("The account handle used when trying to delete
242             account does not exist in system.");
243       }
244
245       for (Post post : allPosts) {
246         if (post.getAuthor().equals(deleteAccount)) {
247           for (Endorsement endorsement : post.getAllEndorsements()) {
248             deletePostFromAllPosts(endorsement);
249           }
250           post.empty();
251         }
252       }
253       deleteAccountFromAllAccounts(deleteAccount);
254     }
255
256       /**
257        * This method allows a user to change their account handle.
258      *
259      * @param oldHandle    The handle of the user already in the system.
260      * @param newHandle    The handle the user would like to change to.
261      *
262      * @throws HandleNotRecognisedException  Thrown when the old handle is not recognised in the system.
263      * @throws IllegalHandleException      Thrown when the new handle is already in use by another account.
264      * @throws InvalidHandleException      Thrown when the new handle is over 30 chars, empty,
265                                                 or contains white space.
266        */
267     @Override
268     public void changeAccountHandle(String oldHandle, String newHandle)
269                     throws HandleNotRecognisedException,
270                     IllegalHandleException,
271                     InvalidHandleException {
272
273       Account changeHandle = getAccount(oldHandle);
274
275       for (Account account : allAccounts) {
276         if (account.getHandle() == newHandle) {
277           throw new IllegalHandleException("Handle is already in use in the system. ");
278         }
279       }
280
281       if (changeHandle == null) {
282         throw new HandleNotRecognisedException("The old account handle used does not exist in the system.");
283       }
284
285       changeHandle.changeHandle(newHandle);
```

```
286     }
287
288       /**
289        * This method allows the user to update their account description.
290      *
291      * @param handle    The handle of the account.
292      * @param description  The new description to be displayed.
293      *
294      * @throws HandleNotRecognisedException Thrown when the handle is not recognised as belonging to
295      * an account in the system.
296        */
297     @Override
298     public void updateAccountDescription(String handle, String description)
299                     throws HandleNotRecognisedException {
300
301       Account accountDescrip = getAccount(handle);
302       if (accountDescrip == null) {
303         throw new HandleNotRecognisedException("The account handle used was not recognised by the system.");
304       }
305       accountDescrip.updateDescription(description);
306     }
307
308       /**
309      * This method allows an account selected with the handle to be displayed as a string,
310      * giving all the important info.
311      *
312      * @param handle  The handle of the account to be displayed
313      *
314      * @throws HandleNotRecognisedException Thrown when the handle provided is not in the system.
315      *
316      * @return Returns the account information as a string.
317      */
318     @Override
319     public String showAccount(String handle)
320                   throws HandleNotRecognisedException {
321
322       Account wantedAccount = getAccount(handle);
323
324       if (wantedAccount == null) {
325         throw new HandleNotRecognisedException("The handle used is not recognised in the system");
326       }
327
328       String accountDisplayed = wantedAccount.toString();
329       return accountDisplayed;
330     }
331
332       /**
333        * This method impliments the SocialMediaPlatform method createPost by creating a post
```

```
334      * for the account identified by the given handle with the following message.
335      *
336      * @param  handle  The handle of the account the post is associated with.
337      * @param  message The message associated with the post.
338      *
339      * @throws HandleNotRecognisedException Thrown If the handle does not match to any
340      *                                      account in the system.
341      * @throws InvalidPostException         Thrown if the message is empty or has more than
342      *                                      100 characters.
343      *
344      * @return The sequential ID of the created post.
345        */
346     @Override
347     public int createPost(String handle, String message)
348                 throws HandleNotRecognisedException,
349                 InvalidPostException {
350       Account author = getAccount(handle);
351       Post newPost = new Post(author, message);
352
353       addPostToAllPosts(newPost);
354
355       author.addNumPost();
356
357       return newPost.getID();
358     }
359
360       /**
361        * This method impliments the SocialMediaPlatform method endorsePost by creating an
362      * endorsement post of an existing post, similar to a retweet on Twitter. An
363      * endorsement post is a special post. It contains a reference to the endorsed post
364      * and its message is formatted as:
365      * <p>
366      * <code>"EP@" + [endorsed account handle] + ": " + [endorsed message]</code>
367      * <p>
368      *
369      * @param handle  The handle of the account associated with the endorsement.
370      * @param id    The id of the post associated with the endorsement.
371      *
372      * @throws HandleNotRecognisedException    Thrown when the handle is not associated with a account
373      *                       on the platform.
374      * @throws PostIDNotRecognisedException    Thrown when the post id is not associated with a post on
375      *                       the platform.
376      * @throws NotActionablePostException    Thrown when attempting to act upon a non actionabe post.
377      *
378      * @return The sequencial ID of the created endorsement.
379        */
380     @Override
381     public int endorsePost(String handle, int id)
```

```
382                 throws HandleNotRecognisedException,
383                 PostIDNotRecognisedException,
384                 NotActionablePostException {
385       Account author = getAccount(handle);
386       Post linkedPost = getPost(id);
387
388       if (linkedPost == null) {
389         throw new PostIDNotRecognisedException("Post ID is not associated with Post in platform.");
390       }
391
392       if (!linkedPost.getIsActionable()) {
393         throw new NotActionablePostException("Attempting to endorse a non-actionable post.");
394       }
395
396       Endorsement newEndorsement = new Endorsement(author, linkedPost);
397       linkedPost.addEndorsement(newEndorsement);
398       addPostToAllPosts(newEndorsement);
399       // increment the linked post's author's number of endorsements and posts.
400       linkedPost.getAuthor().addNumEndorse();
401       linkedPost.getAuthor().addNumPost();
402       // incremenet the endorsement authors number of posts.
403       author.addNumPost();
404       return newEndorsement.getID();
405     }
406
407       /**
408      * This method impliments the SocialMediaPlatform method commentPost by creating a comment post
409      * referring to an existing post, similarly to a reply on Twitter. A comment post is a special
410      * post. It contains a reference to the post being commented upon.
411      *
412      * @param handle  The handle of the account associated with the endorsement.
413      * @param id    The id of the post associated with the endorsement.
414      * @param message  The message conatined in the comment.
415      *
416      * @throws HandleNotRecognisedException    Thrown when the handle is not associated with a account
417      *                       on the platform.
418      * @throws PostIDNotRecognisedException    Thrown when the post id is not associated with a post on
419      *                       the platform.
420      * @throws NotActionablePostException    Thrown when attempting to act upon a non actionabe post.
421      * @throws InvalidPostException      Thrown if the message is empty or has more than
422      *                                       100 characters.
423      *
424      * @return The sequencial ID of the created comment.
425      */
426     @Override
427     public int commentPost(String handle, int id, String message)
428                 throws HandleNotRecognisedException,
429                 PostIDNotRecognisedException,
```

```
430                 NotActionablePostException,
431                 InvalidPostException {
432       Account author = getAccount(handle);
433       Post linkedPost = getPost(id);
434
435       if (linkedPost == null) {
436         throw new PostIDNotRecognisedException("Post ID is not associated with Post in platform.");
437       }
438
439       if (!linkedPost.getIsActionable()) {
440         throw new NotActionablePostException("Attempting to comment on a non-actionable post.");
441       }
442
443       Comment newComment = new Comment(author, message, linkedPost);
444       linkedPost.addComment(newComment);
445       addPostToAllPosts(newComment);
446       author.addNumPost();
447       linkedPost.getAuthor().addNumPost();
448       return newComment.getID();
449     }
450
451       /**
452        * This method impliments the deletePost method of the MiniSocialMediaPlatform interface
453      * by removing the post from the platform. When a post is removed, all
454      * its endorsements should be removed as well. All replies to this post should
455      * be updated by replacing the reference to this post by a generic empty post.
456      * <p>
457      * The generic empty post message should be "The original content was removed
458      * from the system and is no longer available.". This empty post is just a
459      * replacement placeholder for the post which a reply refers to. Empty posts
460      * should not be linked to any account and cannot be acted upon, i.e., it cannot
461      * be available for endorsements or replies.
462      * <p>
463      *
464      * @param id  The id of the post to be displayed.
465      *
466      * @throws PostIDNotRecognisedException    Thrown when the post id is not associated with a post on
467      *                       the platform.
468        */
469     @Override
470     public void deletePost(int id)
471                 throws PostIDNotRecognisedException {
472       Post deletePost = getPost(id);
473
474       if (deletePost == null) {
475         throw new PostIDNotRecognisedException("Post ID of post to be deleted not in the system.");
476       }
477
```

```
478       for (Endorsement endorsement : deletePost.getAllEndorsements()) {
479         // Removing endorsement and decrease the accounts num posts and num endorse
480         deletePostFromAllPosts(endorsement);
481         endorsement.getAuthor().minusNumPost();
482         endorsement.getLinkedPost().getAuthor().minusNumEndorse();
483         endorsement.getLinkedPost().getAuthor().minusNumPost();
484       }
485
486       deletePost.getAuthor().minusNumPost();
487
488       if (deletePost.getClass().getName().startsWith("socialmedia.Endorsement")) {
489         ((Endorsement) deletePost).getLinkedPost().getAuthor().minusNumEndorse();
490         ((Endorsement) deletePost).getLinkedPost().getAuthor().minusNumPost();
491       }
492       else if (deletePost.getClass().getName().startsWith("socialmedia.Comment")) {
493         ((Comment) deletePost).getLinkedPost().getAuthor().minusNumPost();
494       }
495
496       deletePost.empty();
497     }
498
499       /**
500        * This method impliments the showIndividualPost method by finding the Post which coresponds with the
501      * given post ID and the getting the descirption from the post's toString method. The format is
502      * as follows:
503      *
504      * <pre>
505      * ID: [post ID]
506      * Account: [account handle]
507      * No. endorsements: [number of endorsements received by the post] |
508      * No. comments: [number of comments received by the post]
509      * [post message]
510      * </pre>
511      *
512      * @param id    The id of the post to be displayed.
513      *
514      * @throws PostIDNotRecognisedException    Thrown when the post id is not associated with a post on
515      *                       the platform.
516      *
517      * @return The description of the Post.
518        */
519     @Override
520     public String showIndividualPost(int id)
521                     throws PostIDNotRecognisedException {
522       Post wantedPost = getPost(id);
523
524       if (wantedPost == null) {
525         throw new PostIDNotRecognisedException("Post with given id does not exist in system.");
```

```
526       }
527
528       String description = wantedPost.toString();
529
530       return description;
531     }
532
533       /**
534        * This method impliments the showPostChildrenDetails of the MiniSocialMediaPlatform
535      * interface by building a StringBuilder showing the details of the current post and
536      * all its children posts. See interface for further details.
537      *
538      * @param id  The id of the post to be shown.
539      *
540      * @throws PostIDNotRecognisedException  Thrown if the ID does not match to any post in
541      *                                        the system.
542      * @throws NotActionablePostException    Thrown if the ID refers to an endorsement post.
543      *                                        Endorsement posts do not have children
544      *                                        since they are not endorsable nor
545      *                                        commented.
546      *
547      * @return A formatted StringBuilder containing the details of the post and its
548      *         children.
549        */
550     @Override
551     public StringBuilder showPostChildrenDetails(int id)
552                           throws PostIDNotRecognisedException,
553                           NotActionablePostException {
554       StringBuilder postDetails = new StringBuilder();
555
556       Post wantedPost = getPost(id);
557
558       if (wantedPost == null) {
559         throw new PostIDNotRecognisedException("Post with given id does not exist in system.");
560       }
561
562       if (!wantedPost.getIsActionable()) {
563         throw new NotActionablePostException("Attempting to view non-actionable post.");
564       }
565
566       // Add the parent post to the string builder
567       postDetails.append(showIndividualPost(id));
568       postDetails.append("\n | ");
569       // Start the recursive loop over the parent post's children posts
570       addPostChildrenDetails(postDetails, wantedPost.getAllComments(), 0);
571
572       return postDetails;
573     }
```

```
574
575     /**
576      * This is method which uses recurssion to display the posts and all the sub-posts.
577      *
578      * @param postDetails  The StringBuilder to add the posts details to.
579      * @param comments    The comments to loop over adding them to the BuildString
580      *            and recursively calling the method again for any subcomments.
581      * @param numTabs    The number of tabs to add for this set of comments.
582      *
583      * @throws PostIDNotRecognisedException  Throwm if the ID does not match to any post in
584      *                                       the system.
585      */
586     public void addPostChildrenDetails(StringBuilder postDetails, Comment[] comments, int numTabs)
587                     throws PostIDNotRecognisedException {
588       for (Comment comment : comments) {
589         String tabs = new String(new char[numTabs]).replace("\0", "     ");
590         postDetails.append("\n"+tabs+" | > "+showIndividualPost(comment.getID()).replace("\n", "\n     "+tabs));
591         // Check if the comment itself has replies (children comments).
592         if (comment.getAllComments().length != 0) {
593           tabs = new String(new char[numTabs + 1]).replace("\0", "     ");
594           postDetails.append("\n"+tabs+" | ");
595           addPostChildrenDetails(postDetails, comment.getAllComments(), numTabs+1);
596         }
597         // check if the comment news a newline spacing after previous comment.
598         else if (numTabs == 0){
599           postDetails.append("\n");
600         }
601
602       }
603     }
604
605       /**
606        * This method is an implimentation of the getNumberAccounts method in the
607      * SocialMediaPlatform interface by returning the current total number of accounts
608      * present in the platform. Note, this is NOT the total number of accounts ever
609      * created since the current total should discount deletions.
610      *
611      * @return The total number of accounts in the platform.
612        */
613     @Override
614     public int getNumberOfAccounts() {
615       int numAccounts = allAccounts.length;
616       return numAccounts;
617     }
618
619       /**
620        * This method is an implimentation of the getTotalOriginalPosts method in the
621      * SocialMediaPlatform interface by returning the current total number of original
```

```
622      * posts (i.e., disregarding endorsements and comments) present in the platform.
623      * Note, this is NOT the total number of posts ever created since the current total
624      * should discount deletions.
625      *
626      * @return The total number of original posts in the platform.
627        */
628     @Override
629     public int getTotalOriginalPosts() {
630       int numoriginalPosts = 0;
631       for (Post post : allPosts) {
632         if (post.getClass().getName().startsWith("socialmedia.Post")) {
633           numoriginalPosts++;
634         }
635       }
636       return numoriginalPosts;
637     }
638
639       /**
640        * This method is an implimentation of the getTotalEndorsementPost method in the
641      * SocialMediaPlatform interface by returning the current total number of endorsement
642      * posts present in the platform. Note, this is NOT the total number of endorsements
643      * ever created since the current total should discount deletions.
644      *
645      * @return The total number of endorsement posts in the platform.
646        */
647     @Override
648     public int getTotalEndorsmentPosts() {
649       int numEndorsementPosts = 0;
650       for (Post post : allPosts) {
651         if (post.getClass().getName().startsWith("socialmedia.Endorsement")) {
652           numEndorsementPosts++;
653         }
654       }
655       return numEndorsementPosts;
656     }
657
658       /**
659        * This method is an implimentation of the getTotalCommentPosts method in the
660      * SocialMediaPlatform interface by returning the current total number of comments posts
661      * present in the platform. Note, this is NOT the total number of comments ever created
662      * since the current total should discount deletions.
663      *
664      * @return The total number of comments posts in the platform.
665        */
666     @Override
667     public int getTotalCommentPosts() {
668       int numCommentPosts = 0;
669       for (Post post : allPosts) {
```

```
670         if (post.getClass().getName().startsWith("socialmedia.Comment")) {
671           numCommentPosts++;
672         }
673       }
674       return numCommentPosts;
675     }
676 
677       /**
678        * This method implients the getMostEndorsedPost method in the MiniSocialMediaPlatform
679      * iterface by identifing and returning the post with the most number of endorsements,
680      * a.k.a. the most popular post.
681      * <p>
682      * If the method returns -1 then it means there are no Posts in the platform.
683      *
684      * @return The ID of the most popular post.
685        */
686     @Override
687     public int getMostEndorsedPost() {
688       Post mostEndorsed = null;
689       int maxNumEndorsements = 0;
690       int postNumEndorsements;
691       int mostEndorsedId = -1;
692       for (Post post : allPosts) {
693         postNumEndorsements = post.getAllEndorsements().length;
694         if (postNumEndorsements >= maxNumEndorsements) {
695           mostEndorsed = post;
696           maxNumEndorsements = postNumEndorsements;
697         }
698       }
699       if (mostEndorsed != null) {
700         mostEndorsedId = mostEndorsed.getID();
701       }
702 
703       return mostEndorsedId;
704     }
705 
706     /**
707      * This method allows the user to retrieve the account with most endorsements. It loops over every
708      * account within the system and checks over all their posts to see how many endorsements each user
709      * has.
710      * <p>
711      * If the method returns -1 then it means there are no Accounts in the platform.
712      *
713      * @return It returns the Id of the account with the most endorsements on the platform.
714      */
715     @Override
716     public int getMostEndorsedAccount() {
717       Account mostEndorsedAccount = null;
```

```
718       int mostEndorsedAccountId = -1;
719       int maxNumEndorsements = 0;
720       int numEndorse;
721
722       for (Account account : allAccounts) {
723         numEndorse = account.getNumEndorsements();
724         if (numEndorse > maxNumEndorsements) {
725           mostEndorsedAccount = account;
726           maxNumEndorsements = numEndorse;
727         }
728       }
729       if (mostEndorsedAccount != null) {
730         mostEndorsedAccountId = mostEndorsedAccount.getId();
731       }
732       return mostEndorsedAccountId;
733     }
734
735       /**
736        * This method impliments the erasePlatform method in the MiniSocialMediaPlatform
737      * interface by emptying this SocialMediaPlatform of its contents and resets all
738      * internal counters in Account and Post.
739        */
740     @Override
741     public void erasePlatform() {
742       allAccounts = new Account[] {};
743       allPosts = new Post[] {};
744       Post.resetNumPosts();
745       Account.resetNumAccounts();
746     }
747
748       /**
749        * This method impliments the savePlatform method in the MiniSocialMediaPlatform
750      * interace by saving this SocialMediaPlatform's contents into a serialised file, with
751      * the filename given in the argument.
752      *
753      * @param filename The path and name of the file to be saved to.
754      *
755      * @throws IOException Thrown if there is a problem experienced when trying to save the
756      *                     store contents to the file.
757        */
758     @Override
759     public void savePlatform(String filename)
760                 throws IOException {
761       FileOutputStream fileOut = new FileOutputStream(filename);
762       ObjectOutputStream out = new ObjectOutputStream(fileOut);
763       out.writeObject(this.allAccounts);
764       out.writeObject(this.allPosts);
765       out.close();
```

```
766       fileOut.close();
767     }
768
769       /**
770        * This method impliments the loadPlatform method in the MiniSocialMediaPlatform
771      * interface by loading and replacing this SocialMediaPlatform's contents with the
772      * serialised contents stored in the file given in the argument.
773      *
774      * @param filename The location of the file to be loaded.
775      *
776      * @throws IOException            Thrown if there is a problem experienced when trying
777      *                                to load the store contents from the file.
778      * @throws ClassNotFoundException Thrown if required class files cannot be found when
779      *                                loading.
780        */
781     @Override
782     public void loadPlatform(String filename)
783                 throws IOException,
784                 ClassNotFoundException {
785       FileInputStream fileIn = new FileInputStream(filename);
786       ObjectInputStream in = new ObjectInputStream(fileIn);
787       allAccounts = (Account[]) in.readObject();
788       allPosts = (Post[]) in.readObject();
789       in.close();
790       fileIn.close();
791     }
792   }
```