

# Object oriented Programming (ECM1410) CA3

## Printout

March 2021

## 1 Classes

### 1.1 Account

---

```
1 package socialmedia;
2
3 import java.io.Serializable;
4 import java.util.HashSet;
5 import java.util.Set;
6
7 /**
8  * This class contains all the methods and attributes of the Account and therefore
9  * is responsible for handling all the details related to a user on the Social Media
10 * Platform.
11 * <p>
12 * An Account has a unique ID (sequentially incrementing according to num
13 * of users on the system), a string handle (username) and a description similar to
14 * a bio.
15 * <p>
16 * The class itself has a static value for its serialVersionUID used when
17 * serializing the platform, a num of accounts which tracks the number of accounts
18 * created and a HashSet of all usernames (handles).
19 *
20 * @author 700008432
21 * @author 690033172
22 * @version 1.0
23 */
24 public class Account implements Serializable {
25
26     /**
27      * The serialVersionUID which represents the class version.
28      */
29     private static final long serialVersionUID = 8225229775036968396L;
30
```

```

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 IllegalArgumentException 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 IllegalArgumentException,
82             InvalidHandleException{
83         if (!usernames.add(handle)) {
84             throw new IllegalArgumentException("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 IllegalArgumentException  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 IllegalArgumentException,
113             InvalidHandleException {
114         if (!usernames.add(handle)) {
115             throw new IllegalArgumentException("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

---

```
1 package socialmedia;
2
3 import java.io.Serializable;
4
5 /**
6  * This class consists of the methods and attributes of a Post and therefore is
7  * responsible for all the details regarding a Post and due to how a Comment and
8  * Endorsement inherit from Post it is responsible for some methods and details of
9  * a Comment and Endorsement.
10 * <p>
11 * A Post has a unique sequential ID, a message up to 100 characters long, an
12 * author (Account) it is associated with, a list of all comments it receives
13 * and a list of all endorsements it receives.
14 * <p>
15 * The class itself has two static values which are the serialVersionUID which is
16 * used when serializing the platform and numPosts which tracks the number of posts
17 * which have been created.
18 *
19 * @author 700008432
20 * @author 690033172
21 * @version 1.0
22 */
23 public class Post implements Serializable {
24
25     /**
26      * The serialVersionUID which represents the class version.
27      */
28     private static final long serialVersionUID = -6471649759148628316L;
29
30     /**
31      * This is the unique sequential identifier of the post.
32      */
33     private int id;
34
35     /**
36      * This is the message with up to 100 characters.
37      */
38     private String message;
39
40     /**
41      * This is the Account associated with the post.
42      */
43     private Account author;
44
45     /**
```

```

46     * This is an array of all the comments the post receives.
47     */
48     private Comment[] allComments = {};
49
50     /**
51      * This is an array of all the endorsements the post receives.
52     */
53     private Endorsement[] allEndorsements = {};
54
55
56     private boolean isActionable;
57
58     /**
59      * This is a static integer value which records the number of posts in the system starting
60      * at 0, it is used to get the next sequential 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 message with the string "The original content was removed from the system and is no longer
150  * available." and making the post non-actionable.
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 inherited 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 therefore 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 been added on.
39      *
40      * @throws InvalidPostException      Thrown when the message is longer than 100 characters.
41      * @throws HandleNotRecognisedException Thrown when the given account handle 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

---

```
1 package socialmedia;
2
3 /**
4  * This class consists of the specific methods and attributes of a Endorsement, some of the
5  * other methods and attributes of a Endorsement are inherited from a Post. Therefore this
6  * class is responsible for only the methods and attributes unique for an Endorsement.
7  * <p>
8  * As an Endorsement inherits from a Post and therefore has access to all of a Post's attributes
9  * and methods. In addition to these, an Endorsement has a linkedPost attribute which refers
10 * to the Post which the endorsement has been placed on and replicates the message of the post
11 * it is associated with.
12 * <p>
13 * The class itself also has a static value for its serialVersionUID used when
14 * serializing the platform.
15 *
16 * @author 700008432
17 * @author 690033172
18 * @version 1.0
19 */
20 public class Endorsement extends Post {
21
22     /**
23      * The serialVersionUID which represents the class version.
24      */
25     private static final long serialVersionUID = 8178673147762741699L;
26
27     /**
28      * This is the post the comment is associated with.
29      */
30     private Post linkedPost;
31
32     /**
33      * This is the constructor method for an Endorsement, it takes in the parameters author which refers
34      * to the account associated with the post and a Post which the endorsement is associated with.
35      * The message of the post it is associated with is formatted as:
36      *
37      * <p>
38      * <code>"EP@" + [endorsed account handle] + ": " + [endorsed message]</code>
39      * <p>
40      *
41      * @param author      The account associated with the endorsement.
42      * @param linkedPost  The Post which the endorsement has been added on.
43      *
44      * @throws HandleNotRecognisedException Thrown if an Account with given handle does not exist.
45      */
46 }
```

```
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

---

```
1 package socialmedia;
2
3 import java.io.FileInputStream;
4 import java.io.FileOutputStream;
5 import java.io.IOException;
6 import java.io.ObjectInputStream;
7 import java.io.ObjectOutputStream;
8
9 /**
10  * This class is an implementation of the SocialMediaPlatform interface which extends the
11  * MiniSocialMediaPlatform interface. Therefore it is responsible for most of the functionality
12  * of the platform.
13  * <p>
14  * A SocialMedia object has a list of all accounts and a list of all posts which is used to hold
15  * all data on the Social Media Platform. The accounts, posts, endorsements and comments in these
16  * lists are then created, removed and edited using the methods in the class.
17  * <p>
18  * The class itself also has a static value for its serialVersionUID used when
19  * serializing the platform.
20  *
21  * @author 700008432
22  * @author 690033172
23  * @version 1.0
24  */
25 public class SocialMedia implements SocialMediaPlatform {
26
27     /**
28      * The serialVersionUID which represents the class version.
29      */
30     private static final long serialVersionUID = -5306444784233657143L;
31
32     /**
33      * This is an array of all accounts on the platform.
34      */
35     private Account[] allAccounts = {};
36
37     /**
38      * This is an array of all posts on the platform.
39      */
40     private Post[] allPosts = {};
41
42     /**
43      * This method is used to add an instance of an account to the array of all accounts on
44      * the platform.
45      *
```



```

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 IllegalArgumentException 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 IllegalArgumentException,
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 IllegalArgumentException 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 IllegalArgumentException,
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("Attempting 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      * This method allows the user to remove an account using the account handle.
229      *
230      *
231      * @param handle The handle of the account to be removed.
232      *
233      * @throws HandleNotRecognisedException Thrown 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 account
242             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
295      * to 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 sequential 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     linkedPost.getAuthor().addNumEndorse();
400     linkedPost.getAuthor().addNumPost();
401     author.addNumPost();
402     return newEndorsement.getID();
403 }
404
405 /**
406  * This method impliments the SocialMediaPlatform method commentPost by creating a comment post
407  * referring to an existing post, similarly to a reply on Twitter. A comment post is a special
408  * post. It contains a reference to the post being commented upon.
409  *
410  * @param handle The handle of the account associated with the endorsement.
411  * @param id The id of the post associated with the endorsement.
412  * @param message The message conatined in the comment.
413  *
414  * @throws HandleNotRecognisedException Thrown when the handle is not associated with a account
415  * on the platform.
416  * @throws PostIDNotRecognisedException Thrown when the post id is not associated with a post on
417  * the platform.
418  * @throws NotActionablePostException Thrown when attempting to act upon a non actionabe post.
419  * @throws InvalidPostException Thrown if the message is empty or has more than
420  * 100 characters.
421  *
422  * @return The sequencial ID of the created comment.
423  */
424 @Override
425 public int commentPost(String handle, int id, String message)
426     throws HandleNotRecognisedException,
427     PostIDNotRecognisedException,
428     NotActionablePostException,
429     InvalidPostException {

```



```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

766      *
767      * @param filename The location of the file to be loaded.
768      *
769      * @throws IOException          Thrown if there is a problem experienced when trying
770      *                               to load the store contents from the file.
771      * @throws ClassNotFoundException Thrown if required class files cannot be found when
772      *                               loading.
773      */
774      @Override
775      public void loadPlatform(String filename)
776          throws IOException,
777          ClassNotFoundException {
778          FileInputStream fileIn = new FileInputStream(filename);
779          ObjectInputStream in = new ObjectInputStream(fileIn);
780          allAccounts = (Account[]) in.readObject();
781          allPosts = (Post[]) in.readObject();
782          in.close();
783          fileIn.close();
784      }
785  }

```

---