SocialMedia.java

```
1
```

```
1  package socialmedia;
2
3  import java.util.ArrayList;
4  import java.util.Arrays;
5  import java.io.*;
6
7  /**
8   * Implementor of the SocialMediaPlatform interface.
9   */
10 public class SocialMedia implements SocialMediaPlatform {
11     /**
12      * The list of accounts on the platform.
13      */
14     private ArrayList<Account> accountList = new ArrayList<Account>();
15     /**
16      * The list of generic empty posts on the platform.
17      */
18     private ArrayList<Post> emptyPostList = new ArrayList<Post>();
19
20     @Override
21     public int createAccount(String handle) throws IllegalHandleException, In-
22 validHandleException {
23         Account.validateHandle(handle, accountList);
24         Account newAccount = new Account(handle);
25         int numOfAccounts = getNumberOfAccounts();
26         accountList.add(newAccount);
27         assert (numOfAccounts + 1 == getNumberOfAccounts()) : "Number of ac-
28 counts has not increased.";
29         return newAccount.getID();
30     }
31
32     @Override
33     public int createAccount(String handle, String description) throws Ille-
34 galHandleException, InvalidHandleException {
35         Account.validateHandle(handle, accountList);
36         Account newAccount = new Account(handle, description);
37         int numOfAccounts = getNumberOfAccounts();
38         accountList.add(newAccount);
39         assert (numOfAccounts + 1 == getNumberOfAccounts()) : "Number of ac-
40 counts has not increased.";
41         return newAccount.getID();
42     }
43
44     @Override
```

```
45      public void removeAccount(int id) throws AccountIDNotRecognisedException {
46          Account accountToDelete = Account.findAccountById(id, accountList);
47          int numOfAccounts = getNumberOfAccounts();
48          while (accountToDelete.getPosts().size()>0){ // Deletes all posts from
49  the account
50              try{
51                  Post p = accountToDelete.getPosts().get(0);
52                  deletePost(p.getID());
53              } catch(PostIDNotRecognisedException e){
54                  continue;
55              }
56          }
57          accountList.remove(accountToDelete);
58          assert (numOfAccounts - 1 == getNumberOfAccounts()) : "Number of ac-
59  counts has not decreased.";
60      }
61
62      @Override
63      public void removeAccount(String handle) throws HandleNotRecognisedExcep-
64  tion {
65          Account accountToDelete = Account.findAccountByHandle(handle, ac-
66  countList);
67          try{
68              removeAccount(accountToDelete.getID());
69          } catch (AccountIDNotRecognisedException e){
70
71          }
72      }
73
74      @Override
75      public void changeAccountHandle(String oldHandle, String newHandle)
76              throws HandleNotRecognisedException, IllegalHandleException, In-
77  validHandleException {
78          Account.validateHandle(newHandle, accountList);
79          Account account = Account.findAccountByHandle(oldHandle, accountList);
80          account.setHandle(newHandle);
81          assert (account.getHandle() == newHandle) : "Handle has not updated.";
82
83      }
84
85      @Override
86      public void updateAccountDescription(String handle, String description)
87  throws HandleNotRecognisedException {
88          Account account = Account.findAccountByHandle(handle, accountList);
89          account.setDescription(description);
90          assert (account.getDescription() == description):"Description has not
91  updated.";
92      }
```

```
93
94      @Override
95      public String showAccount(String handle) throws HandleNotRecognisedExcep-
96  tion {
97          Account account = Account.findAccountByHandle(handle, accountList);
98          return account.toString();
99      }
100
101     @Override
102     public int createPost(String handle, String message) throws HandleNotRec-
103 ognisedException, InvalidPostException {
104         Account postingAccount = Account.findAccountByHandle(handle, ac-
105 countList);
106         Post.validateMessage(message);
107         Post newPost = new Post(postingAccount, message);
108         int numOfAccountPosts = postingAccount.getPosts().size();
109         postingAccount.addPost(newPost);
110         assert (numOfAccountPosts + 1 == postingAccount.get-
111 Posts().size()):"Account post count not updated.";
112
113         return newPost.getID();
114     }
115
116     @Override
117     public int endorsePost(String handle, int id)
118             throws HandleNotRecognisedException, PostIDNotRecognisedException,
119 NotActionablePostException {
120             Account postingAccount = Account.findAccountByHandle(handle, ac-
121 countList);
122             Post postToEndorse = Post.findPostByID(id, accountList);
123             if (postToEndorse instanceof EndorsementPost){ // Cannot endorse
124 an endorsement post
125                 throw new NotActionablePostException();
126             }
127
128             String message = "EP@" + postToEndorse.getAccount().getHandle() +
129 ": " + postToEndorse.getMessage();
130             int numOfEndorsements = getTotalEndorsmentPosts();
131             EndorsementPost endorsementPost = new EndorsementPost(postingAc-
132 count, message, postToEndorse);
133             assert (postToEndorse.getEndorsements().contains(endorsement-
134 Post)):"Endorsement post not added to list of endorsements.";
135             assert (numOfEndorsements + 1 == getTotalEndorsmentPosts()):"Num-
136 ber of endorsement posts has not increased.";
137             return endorsementPost.getID();
138     }
139
140     @Override
```

```
141     public int commentPost(String handle, int id, String message) throws
142 HandleNotRecognisedException,
143             PostIDNotRecognisedException, NotActionablePostException, In-
144 validPostException {
145 
146         Account postingAccount = Account.findAccountByHandle(handle, ac-
147 countList);
148         Post commentedPost = Post.findPostByID(id, accountList);
149         Post.validateMessage(message);
150 
151         if (commentedPost instanceof EndorsementPost){ //Cannot comment on an
152 endorsement post
153             throw new NotActionablePostException();
154         }
155 
156         int numOfComments = getTotalCommentPosts();
157         Comment newComment = new Comment(postingAccount, message, comment-
158 edPost);
159         assert (commentedPost.getComments().contains(newComment)):"Comment
160 post not added to comment list.";
161         assert (numOfComments + 1 == getTotalCommentPosts()):"Number of com-
162 ment posts has not increased.";
163         return newComment.getID();
164     }
165 
166     @Override
167     public void deletePost(int id) throws PostIDNotRecognisedException {
168         Post postToDelete = Post.findPostByID(id, accountList);
169         if (postToDelete instanceof EndorsementPost){ //Removes the endorse-
170 ment from the post that is endorsed
171             ((EndorsementPost)postToDelete).getReferencePost().removeEndorse-
172 ment((EndorsementPost)postToDelete);
173         }
174         else{
175             postToDelete.clearEndorsements();
176             if (!(postToDelete instanceof Comment)){ // If post is an original
177 post, account post count must be recalculated
178                 postToDelete.getAccount().setPostCountUpToDateToFalse();
179             }
180         }
181         postToDelete.getAccount().setEndorsementCountUpToDateToFalse();
182         postToDelete.getAccount().removePost(postToDelete);
183         postToDelete.setPostToEmpty();
184         emptyPostList.add(postToDelete);
185 
186     }
187 
188     @Override
```

```
189     public String showIndividualPost(int id) throws PostIDNotRecognisedExcep-
190 tion {
191         Post postToShow = Post.findPostByID(id, accountList, emptyPostList);
192         return postToShow.toString();
193     }
194 
195 
196     private StringBuilder DFSChildren(Post originalPost, Post post, int in-
197 dent, StringBuilder sb, ArrayList<Post> visited) throws PostIDNotRecognisedEx-
198 ception{
199         visited.add(post); // List of posts that have been visited by search
200 
201         if(post instanceof Comment){
202             boolean isFirstComment;
203             if (((Comment) post).getReferencePost().getCom-
204 ments().get(0).equals(post)){ //Checks whether post is the first comment under
205 a post
206                 isFirstComment = true;
207             } else{
208                 isFirstComment = false;
209             }
210             sb.append(Post.formatMessage(showIndividualPost(post.getID()), in-
211 dent, isFirstComment));
212         }
213         else{ //Post is an original post
214             sb.append(showIndividualPost(post.getID())+"\n");
215         }
216         indent++;
217 
218         for(Comment c:post.getComments()){ //Recursively calls on all comments
219 under the current post
220             if(!visited.contains(c)){
221                 DFSChildren(originalPost, c, indent, sb, visited);
222             }
223         }
224         return sb;
225     }
226 

227     @Override
228     public StringBuilder showPostChildrenDetails(int id)
229             throws PostIDNotRecognisedException, NotActionablePostException {
230         StringBuilder sb = new StringBuilder();
231         Post postToShow = Post.findPostByID(id, accountList, emptyPostList);
232 
233         if (postToShow instanceof EndorsementPost){ //Cannot call method on
234 endorsement posts
235             throw new NotActionablePostException();
236         }
```

```
237             return DFSChildren(postToShow, postToShow, 0,sb,new Ar-
238     rayList<Post>()); //recursively visits all comments under original post
239         }
240 
241         @Override
242         public int getNumberOfAccounts() {
243             return accountList.size();
244         }
245 
246         @Override
247         public int getTotalOriginalPosts() {
248             int total = 0;
249             for(Account a:accountList){
250                 total += a.getOriginalPostCount();
251             }
252             return total;
253         }
254 
255         @Override
256         public int getTotalEndorsmentPosts() {
257             int total = 0;
258             for(Account a:accountList){
259                 total += a.getEndorsementCount();
260             }
261             return total;
262         }
263 
264         @Override
265         public int getTotalCommentPosts() {
266             int total = 0;
267             for(Account a:accountList){
268                 for(Post p:a.getPosts()){
269                     total += p.getComments().size();
270                 }
271             }
272             return total;
273         }
274 
275         @Override
276         public int getMostEndorsedPost() {
277             Post mostEndorsedPost = null;
278             for(Account a:accountList){
279                 for(Post p : a.getPosts()){
280                     if(mostEndorsedPost == null || p.getNumEndorsements() >
281     mostEndorsedPost.getNumEndorsements()){
282                         mostEndorsedPost = p;
283                     }
284                 }
```

```
285             }
286             if (mostEndorsedPost == null){ // If there are no posts on the plat-
287     form
288                 return 0;
289             }
290             return mostEndorsedPost.getID();
291         }
292 
293         @Override
294         public int getMostEndorsedAccount() {
295             Account mostEndorsedAccount = null;
296             for(Account a : accountList){
297                 if(mostEndorsedAccount == null || a.getEndorsementCount() >
298     mostEndorsedAccount.getEndorsementCount()){
299                     mostEndorsedAccount = a;
300                 }
301             }
302             if (mostEndorsedAccount == null){ // If there are no accounts on the
303     platform
304                 return 0;
305             }
306             return mostEndorsedAccount.getID();
307         }
308 
309         @Override
310         public void erasePlatform() {
311             accountList.clear();
312             emptyPostList.clear();
313             Post.resetIdCount();
314             Account.resetIdCount();
315             assert (accountList.size() == 0) : "Account list not empty";
316         }
317         @Override
318         public void savePlatform(String filename) throws IOException {
319             try(ObjectOutputStream out = new ObjectOutputStream(new FileOut-
320     putStream(filename))){
321                 Account[] accountArr = accountList.toArray(new Account[ac-
322     countList.size()]); //Create an array of accountList
323                 out.writeObject(accountArr);
324                 Post[] emptyPostArr = emptyPostList.toArray(new Post[empty-
325     PostList.size()]); //Create an array of emptyPostList
326                 out.writeObject(emptyPostArr);
327 
328                 int nextAccountID = Account.getNextId(); // Saves account ID coun-
329     ters' value
330                 out.writeObject(nextAccountID);
331                 int nextPostID = Post.getNextId(); // Saves post ID counters'
332     value
```

```
333             out.writeObject(nextPostID);
334         }
335 
336         File f = new File(filename);
337         assert (f.isFile()) : "File has not been created";
338     }
339 
340     @Override
341     public void loadPlatform(String filename) throws IOException, ClassNot-
342 FoundException {
343         try(ObjectInputStream in = new ObjectInputStream(new FileIn-
344 putStream(filename))){
345             Account[] accountArr = (Account[])in.readObject(); // Read as Ar-
346 ray
347             accountList = new ArrayList<>(Arrays.asList(accountArr)); // Cast
348 to ArrayList
349 
350             Post[] emptyPostArr = (Post[])in.readObject();
351             emptyPostList = new ArrayList<>(Arrays.asList(emptyPostArr));
352 
353             int nextAccountID = (int)in.readObject();
354             Account.setNextId(nextAccountID);
355 
356             int nextPostID = (int)in.readObject();
357             Post.setNextId(nextPostID);
358 
359 
360         }
361     }
362 }
```

Account.java

```java
package socialmedia;

import java.util.ArrayList;

import java.io.Serializable;

/**
 * Represents an account on the platform.
 */
public class Account implements Serializable{
    /**
     * ID assigned to next account to be created
     */
    private static int nextID = 1;
    /**
     * The handle associated with the account
     */
    private String handle;
    /**
     * The description associated with the account
     */
    private String description;
    /**
     * The unique integer ID associated with the account
     */
    private int accountID;
    /**
     * Number of original posts posted by the account
     */
    private int postCount = 0;
    /**
     * Number of endorsements refering to a post posted by this account
     * */
    private int endorsementCount = 0;
    /**
     * A flag to indicate whether the postCount attribute is up to date or
should be recalculated
     */
    private boolean postCountUpToDate = false;
    /**
     * A flag to indicate whether the endorsement attribute is up to date or
should be recalculated
     */
    private boolean endorsementCountUpToDate = false;
    /**
```

```
46          * A constant thats value indicates the maximum number of characters a
47      handle can be
48          */
49         private final static int MAX_HANDLE_LENGTH = 30;
50         /**
51          * All posts posted by the account (original, comments and endorsements).
52          */
53         private ArrayList<Post> accountPosts = new ArrayList<Post>();
54
55         /**
56          * Creates an instance of an Account object and calls the overloaded con-
57      structor with the handle and an empty description.
58          * @param handle The handle of the account to be created.
59          */
60         public Account(String handle){
61             this(handle, "");
62
63         }
64         /**
65          * Creates an instance of an account object by assigning the handle and
66      description to the respective parameters. Also, the accountID is assigned
67          * to the next  sequential ID.
68          * @param handle The handle to be associated with the account.
69          * @param description The description to be associated with the account.
70          */
71         public Account(String handle, String description){
72             this.handle = handle;
73             this.description = description;
74             accountID = nextID++;
75         }
76         /**
77          * Returns the handle associated with the account.
78          * @return The handle of the account.
79          */
80         public String getHandle(){
81             return handle;
82         }
83         /**
84          * Returns the decscription associated with the account.
85          * @return The description of the account.
86          */
87         public String getDescription(){
88             return description;
89         }
90         /**
91          * Returns the ID associated with the account.
92          * @return The ID related with the account.
93          */
```

```
94      public int getID(){
95          return accountID;
96      }
97      /**
98       * Returns the number of original posts (excluding comments and endorse-
99  ments) the account has made.
100      * @return The number of original posts on the account.
101      */
102     public int getOriginalPostCount(){
103         if(!postCountUpToDate){
104             int total=0;
105             for(Post p:accountPosts){
106                 if(!(p instanceof Comment || p instanceof EndorsementPost)){
107                     total += 1;
108                 }
109             }
110             postCount = total; // cache the calculated value
111             postCountUpToDate = true;
112         }
113         return postCount;
114     }
115     /**
116      * Returns the number of endorsements for posts that are made for the ac-
117  count.
118      * @return The total number of posts that are endorsing the posts made by
119  the account.
120      */
121     public int getEndorsementCount(){
122         if(!endorsementCountUpToDate){
123             int total = 0;
124             for(Post p:accountPosts){
125                 total += p.getNumEndorsements();
126             }
127             endorsementCount = total; // Cache the calculated value
128             endorsementCountUpToDate = true;
129         }
130         return endorsementCount;
131     }
132     /**
133      * Returns the number of posts made by the account. This includes original
134  posts, comments and endorsements
135      * @return The number of posts made by the account.
136      */
137     public int getTotalPostCount(){
138         return accountPosts.size();
139     }
140     /**
```

```
141         * Returns the ArrayList accountPosts which contains all of the posts cre-
142     ated by the account. This includes
143         * original posts, comments and endorsements.
144         * @return ArrayList of posts created by the account
145         */
146        public ArrayList<Post> getPosts(){
147            return accountPosts;
148        }
149        /**
150         * Returns the next sequential ID that will be assigned to the next ac-
151     count to be created
152         * @return ID to be used by the next account to be created
153         */
154        public static int getNextId(){
155            return nextID;
156        }
157
158        //Setters
159        /**
160         * Sets the next sequential ID, that will be assinged to the next account
161     to be created, to the value
162         * passed in
163         * @param id The value to assign to nextID
164         */
165        public static void setNextId(int id){
166            nextID=id;
167        }
168        /**
169         * Sets the handle of the account to a new value.
170         * @param newHandle The value to change the handle of the account to.
171         */
172        public void setHandle(String newHandle){
173            handle = newHandle;
174        }
175        /**
176         * Sets the description of the account to a new value.
177         * @param newDescription The value to change the description of the ac-
178     count to.
179         */
180        public void setDescription(String newDescription){
181            description = newDescription;
182        }
183        /**
184         * Resets the sequential ID to a value of 1.
185         */
186        public static void resetIdCount(){
187            nextID = 1;
188        }
```

```
189         /**
190          * Set the endorsementCountUpToDate attribute to false, so when the
191     getEndorsementCount() method is called the number of endorsements related to
192          * the account is recalculated.
193          */
194         public void setEndorsementCountUpToDateToFalse(){
195             this.endorsementCountUpToDate = false;
196         }
197         /**
198          * Sets the boolean value of postCountUpToDate to false. postCountUpToDate
199     states whether the count of original posts
200          * is up to date.
201          */
202         public void setPostCountUpToDateToFalse(){
203             this.postCountUpToDate = false;
204         }
205         /**
206          * Adds a post to the list of posts that the account has made.
207          * @param p The post to be added to the list of posts that the account has
208     made.
209          */
210         public void addPost(Post p){
211             this.accountPosts.add(p);
212             if (!(p instanceof Comment || p instanceof EndorsementPost)){ //If
213     post is original post
214                 postCountUpToDate = false;
215             }
216 
217         }
218         /**
219          * Removes the post (specified by parameter p) from the list of posts that
220     the account has made.
221          * @param p The post to be removed from the list of posts that the account
222     has made.
223          */
224         public void removePost(Post p){
225             this.accountPosts.remove(p);
226             if (!(p instanceof Comment || p instanceof EndorsementPost)){ // If
227     post is original post
228                 postCountUpToDate = false;
229             }
230         }
231 
232         /**
233          * Returns a string that contains information about an account including
234     its unique account ID, the handle
235          * of the account, the description of the account, the number of posts the
236     account has created (original posts,
```

```
237         * comments and endorsements) and the number of endorsement posts that re-
238     fer to a post created by this account
239         *
240         * <pre>
241         * ID: 1
242         * Handle: user1
243         * Description: This is my description
244         * Post count: 1
245         * Endorse count: 1
246         * </pre>
247         */
248        @Override
249        public String toString(){
250            return  "ID: " + accountID +
251                    "\nHandle: " + handle +
252                    "\nDescription: " + description +
253                    "\nPost count: " + getTotalPostCount() +
254                    "\nEndorse count: " + getEndorsementCount();
255        }
256        /**
257         * Validates a string to be used as the handle of an account by checking
258     that it is less than 30 characters, not empty, contains
259         * no white space and is not being used by any other account in the sys-
260     tem.
261         * @param handle The string to be used as the handle of an account.
262         * @param accountList An ArrayList of account objects.
263         * @throws InvalidHandleException This is thrown when the handle is more
264     than 30 characters, or is an empty string, or contains any whitespace.
265         * @throws IllegalHandleException This is thrown when an account already
266     contains the handle defined in the parameter handle.
267         */
268        public static void validateHandle(String handle, ArrayList<Account> ac-
269     countList) throws InvalidHandleException, IllegalHandleException{
270            if(handle.length() > MAX_HANDLE_LENGTH || handle.isEmpty() || han-
271     dle.matches("(\\s)+")){ //Checks string is not empty, less than 30 characters
272     and contains no whitespace
273                throw new InvalidHandleException();
274            }
275            for(Account a : accountList){ // Checks if any account already owns
276     this handle
277                if(a.getHandle().equals(handle)){
278                    throw new IllegalHandleException();
279                }
280            }
281        }
282        /**
```

```
283         * Returns the account with the handle defined by parameter handle if it
284     exists in the ArrayList accounts, otherwise a HandleNotRecognisedException is
285     thrown.
286         * @param handle The account with the handle to find.
287         * @param accounts An ArrayList of Account objects.
288         * @return The account object that has the handle that is equal to the pa-
289     rameter handle.
290         * @throws HandleNotRecognisedException This is thrown if no account has a
291     handle defined by the parameter handle in the ArrayList of accounts (parameter
292     accounts).
293         */
294        public static Account findAccountByHandle(String handle, ArrayList<Ac-
295     count> accounts) throws HandleNotRecognisedException{
296            for (Account a : accounts){
297                if (a.getHandle().equals(handle)){
298                    return a;
299                }
300            }
301            throw new HandleNotRecognisedException();
302        }
303        /**
304         * Returns the account with the ID defined by the parameter id if it con-
305     tained in the ArrayList of accounts, otherwise an AccountIDNotRecognisedExcep-
306     tion is thrown.
307         * @param id The ID of the account to find.
308         * @param accounts An ArrayList of Accounts.
309         * @return The account with the ID that is equal to the parameter id.
310         * @throws AccountIDNotRecognisedException This is thrown if no account in
311     the ArrayList of accounts that has an ID that is equal to the parameter id.
312         */
313        public static Account findAccountById(int id, ArrayList<Account> accounts)
314     throws AccountIDNotRecognisedException{
315            for (Account a : accounts){
316                if (a.getID() == id){
317                    return a;
318                }
319            }
320            throw new AccountIDNotRecognisedException();
321        }
322
323    }
  1
```

Post.java

```java
package socialmedia;

import  java.util.ArrayList;

import java.io.Serializable;

/**
 * Represents a post on the platform.
 */
public class Post implements Serializable{
    /**
     * ID assigned to next post to be created
     */
    protected static int nextID = 1;
    /**
     * The account that is making the post.
     */
    protected Account account;
    /**
     * The unique integer ID associated with the post.
     */
    protected int postID;
    /**
     * The text that is associated with the post.
     */
    protected String message;
    /**
     * The list of comments that refer to this post.
     */
    protected ArrayList<Comment> comments = new ArrayList<Comment>();
    /**
     * The list of endorsements that are about this post.
     */
    protected ArrayList<EndorsementPost> endorsements = new ArrayList<Endorse-
mentPost>();
    /**
     * A flag to indicate whether this post is a generic empty post.
     */
    protected boolean isEmptyPost;
    /**
     * The number of comments on this post.
     */
    protected int commentCount;
    /**
     * A flag to indicate whether the commentCount attribute is up to date.
     */
```

```
47      protected boolean commentCountUptoDate;
48
49      /**
50       * Creates an instance of the Post object. Assigns a message, an ID and an
51  account to the post object.
52       * @param account Account that is creating the post
53       * @param message Text contained in the post
54       */
55      public Post(Account account, String message){
56          this.message=message;
57          this.account=account;
58          postID=nextID++;
59          isEmptyPost = false;
60          commentCountUptoDate = false;
61      }
62
63      /**
64       * Returns the account that created the post
65       * @return Account that created the post
66       */
67      public Account getAccount(){
68          return account;
69      }
70      /**
71       * Returns the ID of the post
72       * @return Unique ID of post
73       */
74      public int getID(){
75          return postID;
76      }
77      /**
78       * Returns the text message that the post contains
79       * @return Text that the post contains
80       */
81      public String getMessage(){
82          return message;
83      }
84      /**
85       * Returns the total number of comments directly about the post
86       * @return Number of comments the post has
87       */
88      public int getNumComments(){
89          if (!commentCountUptoDate){
90              int total = 0;
91              for (Comment c : comments){
92                  if (!c.isEmptyPost()){
93                      total +=1;
94                  }
```

```
95                }
96                commentCount = total; // Cache the calculated value
97                commentCountUptoDate = true;
98            }
99            return commentCount;
100       }
101       /**
102        * Sets the commentCoutUpToDate varaible to false
103        */
104       public void setCommentCountUptoDateToFalse(){
105           commentCountUptoDate = false;
106       }
107       /**
108        * Returns the total number of endorsement about the post
109        * @return Number of endorsements about the post
110        */
111       public int getNumEndorsements(){
112           return endorsements.size();
113       }
114
115       /**
116        * Returns the List of comments directly under the post
117        * @return ArrayList of comment objects about the post
118        */
119       public ArrayList<Comment> getComments() {
120           return comments;
121       }
122
123       /**
124        * Returns the List of endorsement posts about the post
125        * @return ArrayList of endorsementPost objects about the post
126        */
127       public ArrayList<EndorsementPost> getEndorsements() {
128           return endorsements;
129       }
130       /**
131        * Returns the next sequential ID that will be assigned to the next post
132   that is created
133        * @return ID to be used by the next post that is created
134        */
135       public static int getNextId(){
136           return nextID;
137       }
138
139       /**
140        * States whether the post is an empty post. An empty post is a post that
141        * has been deleted so no longer has a message
142        * @return Boolean value stating whether the post is empty or not
```

```
143         */
144        public boolean isEmptyPost(){
145            return isEmptyPost;
146        }
147
148        /**
149         * Sets the next sequential ID, that will be assinged to the next post
150    that is created, to the value
151         * passed in
152         * @param id The value to assign to nextID
153         */
154        public static void setNextId(int id){
155            nextID=id;
156        }
157        /**
158         * Updates the message of the post to provided message
159         * @param message New text that the post object shows when printed
160         */
161        public void setMessage(String message){
162            this.message = message;
163        }
164        /**
165         * Sets the post to be an empty post. If a post is deleted, it is set to
166    empty. Its message is updated
167         * to state that it has been deleted. Comments are still linked to the
168    post
169         */
170        public void setPostToEmpty(){
171            this.isEmptyPost = true;
172            this.account = null;
173            this.message = "The original content was removed from the system and
174    is no longer available.";
175        }
176        /**
177         * Restets the static ID count of the Post class so that the next post
178    created will have an ID of 1
179         */
180        public static void resetIdCount(){
181            nextID = 1;
182        }
183        /**
184         * Add a comment to the list of comments about the post
185         * @param c Comment object about the post
186         */
187        public void addComment(Comment c){
188            comments.add(c);
189            commentCountUptoDate = false;
190        }
```

```
191         /**
192          * Add an endorsement post to the list of endorsements about the post
193          * @param e Endorsement object about the post
194          */
195         public void addEndorsementPost(EndorsementPost e){
196             endorsements.add(e);
197         }
198 
199         /**
200          * Removes a single EndorsementPost object from the list of edorsements
201     stored in a post object
202          * @param e The EndorsementPost object that is being removed
203          */
204         public void removeEndorsement(EndorsementPost e){
205             endorsements.remove(e);
206         }
207         /**
208          * Clears the list of endorsements stored in the post Object
209          */
210         public void clearEndorsements(){
211             this.endorsements.clear();
212         }
213 
214         /**
215          * Checks that the message meets the requirements to be posted. The mes-
216     sage must be
217          * less than 100 characters and must not be empty
218          * @param message Text that is being checked
219          * @throws InvalidPostException If the message is empty or if it is longer
220     than 100 characters
221          */
222         public static void validateMessage(String message) throws InvalidPostEx-
223     ception{
224             if (message.length()>100 || message.isEmpty()){
225                 throw new InvalidPostException();
226             }
227         }
228         /**
229          * Searches through all of the posts stored in accounts and all of the
230     empty posts stored in an emptyPostList.
231          * Returns the Post object that has the given ID
232          * @param id ID of the post to be returned
233          * @param accountList List of accounts stored in SocialMedia
234          * @param emptyPostList List of empty posts
235          * @return Post that has the ID that is passed in
236          * @throws PostIDNotRecognisedException There is no post that has the
237     given ID
238          */
```

```
239      public static Post findPostByID(int id, ArrayList<Account> accountList,
240  ArrayList<Post> emptyPostList) throws PostIDNotRecognisedException{
241          for (Account a : accountList){ // Searches through posts stored in ac-
242  counts
243              for(Post p : a.getPosts()){
244                  if (p.getID() == id){
245                      return p;
246                  }
247              }
248          }
249          for (Post p : emptyPostList){ // Searches through posts stored in emp-
250  tyPostList
251              if (p.getID() == id){
252                  return p;
253              }
254          }
255          throw new PostIDNotRecognisedException();
256      }
257      /**
258       * Searches through all of the posts stored in accounts and returns the
259  Post object that has the given ID
260       * @param id ID of the post to be returned
261       * @param accountList List of accounts stored in SocialMedia
262       * @return Post that has the ID that is passed in
263       * @throws PostIDNotRecognisedException There is no post that has the
264  given ID
265       */
266      public static Post findPostByID(int id, ArrayList<Account> accountList)
267  throws PostIDNotRecognisedException{
268          for (Account a : accountList){ // Searches through posts stored in ac-
269  counts
270              for(Post p : a.getPosts()){
271                  if (p.getID() == id){
272                      return p;
273                  }
274              }
275
276          }
277          throw new PostIDNotRecognisedException();
278      }
279
280      /**
281       * Formats the text to be printed by the showPostChildrenDetails method.
282  Adds |> before the first line
283       * and some number of indents before each new line so that the message is
284  indented when printed. If it is
285       * the first comment of a post a blank line containing a single | is added
286  about. So if the text passed
```

```
287         *  in is the first comment of a post it returns:
288         *
289         * <pre>
290         *     |
291         *     | > ID: 1
292         *     Account: user1
293         *     No. endorsements: 0 | No. comments: 0
294         *     This is the first comment
295         * </pre>
296         *
297         * @param message The description of the post that is being formatted
298         * @param indentLevel The number of times the message is indented
299         * @param isFirstComment Boolean value stating whether the comment is the
300     first comment of a post
301         * @return New string which is the indented message with the appropriate
302     symbols | > before the first line
303         */
304        public static StringBuilder formatMessage(String message, int indentLevel,
305     boolean isFirstComment){
306             StringBuilder sb = new StringBuilder();
307             boolean isFirstLine = true;
308             for(String line : message.split("\n")){ //Splits message into individ-
309     ual lines
310                 if(isFirstLine){
311                     String firstLine = "";
312                     if (indentLevel != 0 && isFirstComment){
313                         firstLine += "    ".repeat(indentLevel-1)+ "|\n"; // Add
314     empty line with |
315                     }
316                     sb.append(firstLine+"    ".repeat(indentLevel-1)+"| >
317     "+line+"\n"); //Indent line and add '| >' before first line
318                     isFirstLine = false;
319                 }
320                 else{ // not first line
321                     sb.append("    ".repeat(indentLevel)+line+"\n"); //Indent line
322                 }
323             }
324             return sb;
325         }
326         /**
327          * Returns a string containing the information about a post. This includes
328     the ID of the post, the account
329          * that the post belongs to and the number of endorsements and number of
330     comments that the post has. It will
331          * return a string in the form below.
332          * <pre>
333          * ID: 1
334          * Account: user1
```

```
     * No. endorsements: 2 | No. comments: 1
     * This is a post
     * </pre>
     * <p>
     * An empty post, which has been deleted will display a message stating
that it has been deleted.
     * <pre>
     * ------------------------------------------------------------------------
-----
     * The original content was removed from the system and is no longer
available.
     * ------------------------------------------------------------------------
-----
     * </pre>
     * @return String of information about the post
     */
    @Override
    public String toString(){
        if (isEmptyPost){
            return "-".repeat(message.length())+"\n"+message+"\n"+"-".re-
peat(message.length())+"\n";
        }
        return "ID: "+ postID +
        "\nAccount: "+ account.getHandle()+
        "\nNo. endorsements: "+ getNumEndorsements()+" | No. comments: "+
getNumComments()+ "\n" +
        message;
    }
}
```

Comment.java

```java
package socialmedia;

/**
 * Represents a comment post on the platform.
 */
public class Comment extends Post{
    /**
     * The post object that the comment is under.
     */
    private Post referencePost;

    /**
     * Creates an instance of a Comment object, by first calling the post
     * superclass constructor and then assigning a value for reference post.
     * @param account the account that is creating the comment.
     * @param message the text contained in the comment object.
     * @param referencePost the post object that the comment refers to.
     */
    public Comment(Account account, String message, Post referencePost) {
        super(account, message);
        this.referencePost = referencePost;
        referencePost.addComment(this);
        account.addPost(this);
    }

    /** This returns the post object that the comment refers to.
     * @return Post that the comment refers to.
     */
    public Post getReferencePost(){
        return referencePost;
    }
}
```

EndorsementPost.java

```java
package socialmedia;
/**
 * Represents an endorsement post on the platform.
 */
public class EndorsementPost extends Post{

    /**
     * The post object that the endorsement post is endorsing.
     */
    private Post referencePost;
    /**
     * Creates an instance of a EndorsementPost object, by first calling the post
     * superclass constructor and then assigning a value for reference post.
     * @param account the account that is posting the endorsement.
     * @param message The text shown in the post body. This will contain the original post message
     * @param referencePost The post that is being endorsed
     */
    public EndorsementPost(Account account, String message, Post referencePost) {
        super(account, message);
        this.referencePost = referencePost;
        referencePost.addEndorsementPost(this);
        referencePost.getAccount().setEndorsementCountUpToDateToFalse();//postingAccount.setEndorsementCountUpToDateToFalse();
        account.addPost(this);
    }
    /**
     * Returns the post object that the endorsement referes to
     * @return Post that endorsement referes to
     */
    public Post getReferencePost(){
        return this.referencePost;
    }
}
```