1 SocialMedia.java

```java
package socialmedia;
import java.io.IOException;
import java.util.HashMap;
import java.io.*;
import java.util.ArrayList;




/**
 *  The social media class houses all the methods necessary for loading the social media platform.
 *
 * @author Student Number - 720080382, Candidate Number - 232950
 * @author Student Number - 700022556, Candidate Number - 245071
 * @version 30/03/2023
 */
public class SocialMedia implements SocialMediaPlatform {

   // Account-related methods ***************************************

   /**
    * The method creates an account in the platform with the given handle and
    * description.
    * <p>
    * The state of this SocialMediaPlatform must be be unchanged if any exceptions
    * are thrown.
    *
    * @param handle    account's handle.
    * @param description account's description.
    * @throws IllegalHandleException if the handle already exists in the platform.
    * @throws InvalidHandleException if the new handle is empty, has more than 30
    *                                characters, or has white spaces.
    * @return the ID of the created account.
    */

   @Override
   public int createAccount(String handle, String description) throws IllegalHandleException,
       InvalidHandleException {


      if (Account.doesHandleExist(handle) == true) {
         throw new IllegalHandleException("An account with this handle alreayd exists" + handle); //If the
             proposed handle already exists in the system, an exeption is thrown.
      }

      if (Account.isHandleInvalid(handle) == true) {
         throw new InvalidHandleException("The handle inputted either has a whitespace, is more than 30
             characters, or is empty"); //If the proposed handle does not meet the requirements of the
             system, an exception is thrown.
      }

      Account newAccount = new Account(); //Initialises a new Account object.
```

```
49
50      newAccount.setHandle(handle);
51      newAccount.setDescription(description);
52      newAccount.setAccountId(Account.generateUniqueRandomNumber());
53
54      (Account.accountArrayList).add(newAccount); //Adding the new account to our accountArrayList so we can
            find/use it later.
55
56      return newAccount.getAccountId(); //Returning the ID of the created account.
57
58  }
59
60      /**
61   * The method creates an account in the platform with the given handle.
62   * <p>
63   * The state of this SocialMediaPlatform must be be unchanged if any exceptions
64   * are thrown.
65   *
66   * @param handle account's handle.
67   * @throws IllegalHandleException if the handle already exists in the platform.
68   * @throws InvalidHandleException if the new handle is empty, has more than 30
69   *                                characters, or has white spaces.
70   * @return the ID of the created account.
71   *
72   */
73  @Override
74  public int createAccount(String handle) throws IllegalHandleException, InvalidHandleException {
75
76      if (Account.doesHandleExist(handle) == true) {
77         throw new IllegalHandleException("And account with this handle alreayd exists in the system"); //If
               the proposed handle already exists in the system, an exeption is thrown.
78      }
79
80      if (Account.isHandleInvalid(handle) == true) {
81          throw new InvalidHandleException("The handle inputted is not valid for the system. It either
                contains whitespace, is more than 30 characters or is empty"); //If the proposed handle does
                not meet the requirements of the system, an exception is thrown.
82      }
83
84      Account newAccount = new Account();
85
86      newAccount.setHandle(handle);
87      //This method doesn't parse a description, so we don't use our setter mehtod for an account's
            description.
88
89      newAccount.setAccountId(Account.generateUniqueRandomNumber());
90
91      (Account.accountArrayList).add(newAccount); //Adding the new Account to the system.
92
93      return newAccount.getAccountId();
94
95  }
96      /**
97   * The method removes the account with the corresponding ID from the platform.
98   * When an account is removed, all of their posts and likes will also be
```

```
99      * removed.
100     * <p>
101     * The state of this SocialMediaPlatform must be be unchanged if any exceptions
102     * are thrown.
103     *
104     * @param id ID of the account.
105     * @throws AccountIDNotRecognisedException if the ID does not match to any
106     *                                         account in the system.
107     */
108    @Override
109    public void removeAccount(int id) throws AccountIDNotRecognisedException {
110
111       //The following assertion checks that there are accounts in the system that can be deleted:
112       assert ((Account.accountArrayList).size() > 0) : "There are no accounts in the system to delete.";
113       SocialMedia post = new SocialMedia(); // a new social media object is created so to use the removepost
              late on
114
115
116       //The following block checks if the id actually exists in the system. If it doesn't, we throw
              AccountIDNotRecognisedException.
117       if (Account.isAccountIdRecognised(id) == false) {
118          throw new AccountIDNotRecognisedException("The ID does not match any in the system");
119       }
120
121       for (int i = 0; i < (Account.accountArrayList).size(); i++) {
122          if ((((Account.accountArrayList).get(i)).getAccountId()) == id) {
123             for (int j = 0; j <(Post.postArrayList).size(); j++){
124                if (((Post.postArrayList).get(j)).getAccountHandle() ==
                       (((Account.accountArrayList).get(i)).getHandle())) {
125                   try{post.deletePost(((Post.postArrayList).get(j)).getPostId());}catch(PostIDNotRecognisedException
                          e ){e.printStackTrace();}//Deletes all posts made by the account that is being removed.
126                }
127             }
128             (Account.accountArrayList).remove(i); //Remove the account with the specified Id.
129          }
130       }
131
132
133
134    }
135
136
137    /**
138     * The method removes the account with the corresponding handle from the
139     * platform. When an account is removed, all of their posts and likes should
140     * also be removed.
141     * <p>
142     * The state of this SocialMediaPlatform must be be unchanged if any exceptions
143     * are thrown.
144     *
145     * @param handle account's handle.
146     * @throws HandleNotRecognisedException if the handle does not match to any
147     *                                      account in the system.
148     */
149
```

```
150     @Override
151     public void removeAccount(String handle) throws HandleNotRecognisedException {
152
153        //The following assertion checks that there are accounts in the system that can be deleted:
154        assert ((Account.accountArrayList).size() > 0) : "There are no accounts in the system to delete.";
155        SocialMedia post = new SocialMedia(); // a new social media object is created so to use the removepost
               late on
156
157
158        if (Account.doesHandleExist(handle) == false) {
159           throw new HandleNotRecognisedException("The handle inputted does not match any in the system");
160        }
161
162
163        for (int i = 0; i < (Account.accountArrayList).size(); i++) {
164           if ((((Account.accountArrayList).get(i)).getHandle()).equals(handle)) {
165              for (int j = 0; j <(Post.postArrayList).size(); j++){
166                 if (((Post.postArrayList).get(j)).getAccountHandle() ==
                        (((Account.accountArrayList).get(i)).getHandle())) {
167                    try{post.deletePost(((Post.postArrayList).get(j)).getPostId());}catch(PostIDNotRecognisedException
                           e ){e.printStackTrace();}//Deletes all posts made by the account that is being removed.
168                 }
169
170           }(Account.accountArrayList).remove(i); //Remove the account with the specified Id.
171           }
172        }}
173
174
175        /**
176      * The method replaces the oldHandle of an account by the newHandle.
177      * <p>
178      * The state of this SocialMediaPlatform must be be unchanged if any exceptions
179      * are thrown.
180      *
181      * @param oldHandle account's old handle.
182      * @param newHandle account's new handle.
183      * @throws HandleNotRecognisedException if the old handle does not match to any
184      *                                      account in the system.
185      * @throws IllegalHandleException       if the new handle already exists in the
186      *                                      platform.
187      * @throws InvalidHandleException       if the new handle is empty, has more
188      *                                      than 30 characters, or has white spaces.
189      */
190     @Override
191     public void changeAccountHandle(String oldHandle, String newHandle)
192           throws HandleNotRecognisedException, IllegalHandleException, InvalidHandleException {
193
194              //The following assertion checks that the old handle is being changed to a new, different handle.
195              assert (oldHandle != newHandle) : "The new handle cannot be the same as the old handle.";
196
197
198              //The following block checks if the old handle actually exists in the system. If it doesn't, we
                     throw HandleNotRecognisedException.
199              if (Account.doesHandleExist(oldHandle) == false) {
200                 throw new HandleNotRecognisedException("The oldhandle doesnt match any in the system: " +
```

```
                oldHandle);
201         }
202
203         //The following block checks if the new handle already exists in the system. If it does, we
                throw IllegalHandleException.
204         if (Account.doesHandleExist(newHandle) == true) {
205            throw new IllegalHandleException("The newhandle alreayd exists in the platform: " + newHandle);
206         }
207
208         //The following block checks if the new handle is not empty and does not contain whitespace AND
                is no longer than 30 chars. If it fails this check, we throw InvalidHandleException.
209         if (Account.isHandleInvalid(newHandle) == true) {
210            throw new InvalidHandleException("This handle is invalid");
211          }
212
213
214         for (int i = 0; i < (Account.accountArrayList).size(); i++) {
215            if ((((Account.accountArrayList).get(i)).getHandle()).equals(oldHandle)) {
216               ((Account.accountArrayList).get(i)).setHandle(newHandle);
217            }
218         }
219
220       }
221
222
223
224   /**
225    * The method updates the description of the account with the respective handle.
226    * <p>
227    * The state of this SocialMediaPlatform must be be unchanged if any exceptions
228    * are thrown.
229    *
230    * @param handle    handle to identify the account.
231    * @param description new text for description.
232    * @throws HandleNotRecognisedException if the handle does not match to any
233    *                                  account in the system.
234    */
235   @Override
236   public void updateAccountDescription(String handle, String description) throws
          HandleNotRecognisedException {
237
238      if (Account.doesHandleExist(handle) == false) {
239         throw new HandleNotRecognisedException("the handle inputted does not match any in the system");
240      }
241
242      //    The entire account array is iterated through
243      for (int i = 0; i < (Account.accountArrayList).size(); i++) {
244         if ((((Account.accountArrayList).get(i)).getHandle()).equals(handle)) {
245            ((Account.accountArrayList).get(i)).setDescription(description); //A setter method is used to
                   update the account description
246      }
247   }
248   }
249
250   /**
```

```
251      * The method creates a formatted string summarising the stats of the account
252      * identified by the given handle. The template should be:
253      *
254      * <pre>
255      * ID: [account ID]
256      * Handle: [account handle]
257      * Description: [account description]
258      * Post count: [total number of posts, including endorsements and replies]
259      * Endorse count: [sum of endorsements received by each post of this account]
260      * </pre>
261      *
262      * @param handle handle to identify the account.
263      * @return the account formatted summary.
264      * @throws HandleNotRecognisedException if the handle does not match to any
265      *                                      account in the system.
266      */
267     @Override
268     public String showAccount(String handle) throws HandleNotRecognisedException {
269        String output = "";
270        if (Account.doesHandleExist(handle) == false) {
271           throw new HandleNotRecognisedException("The handle inputted does not match any in the system");
272        }
273
274        for (int i = 0; i < (Account.accountArrayList).size(); i++) {
275           if ((((Account.accountArrayList).get(i)).getHandle()).equals(handle)) {
276              //The output is formatted. The \n is added so to resemble the template
277              output = String.format("ID: %s \nHandle: %s \nDescription: %s \nPost count: %s \nEndorse Count:
                     %s "
                     ,((Account.accountArrayList).get(i)).getAccountId(),((Account.accountArrayList).get(i)).getHandle(),((A
278           }
279        }
280        return output;
281     }
282
283
284     // End Account-related methods ***************************************
285     // Post-related methods ****************************************
286
287     /**
288      * The method creates a post for the account identified by the given handle with
289      * the following message.
290      * <p>
291      * The state of this SocialMediaPlatform must be be unchanged if any exceptions
292      * are thrown.
293      *
294      * @param handle handle to identify the account.
295      * @param message post message.
296      * @throws HandleNotRecognisedException if the handle does not match to any
297      *                                      account in the system.
298      * @throws InvalidPostException     if the message is empty or has more than
299      *                                      100 characters.
300      * @return the sequential ID of the created post.
301      */
302     @Override
303     public int createPost(String handle, String message) throws HandleNotRecognisedException,
```

```
      InvalidPostException {
304
305     if (Account.doesHandleExist(handle) == false) {
306        throw new HandleNotRecognisedException("The handle inputted does not match nay in the system");
307     }else if (Post.isPostInvalid(message) == true) {
308        throw new InvalidPostException("Either the message is empty or is more than a hundred characters");
309     }
310
311
312     Post newPost = new Post(handle, message);
313     (Post.postArrayList).add(newPost); // the post is created using the constructor above and then added
          to our array of posts
314     return newPost.getPostId();
315
316
317
318
319
320
321     }
322
323
324
325   /**
326    * The method creates an endorsement post of an existing post, similar to a
327    * retweet on Twitter. An endorsement post is a special post. It contains a
328    * reference to the endorsed post and its message is formatted as:
329    * <p>
330    * <code>"EP@" + [endorsed account handle] + ": " + [endorsed message]</code>
331    * <p>
332    * The state of this SocialMediaPlatform must be be unchanged if any exceptions
333    * are thrown.
334    *
335    * @param handle of the account endorsing a post.
336    * @param id    of the post being endorsed.
337    * @return the sequential ID of the created post.
338    * @throws HandleNotRecognisedException if the handle does not match to any
339    *                                      account in the system.
340    * @throws PostIDNotRecognisedException if the ID does not match to any post in
341    *                                      the system.
342    * @throws NotActionablePostException if the ID refers to a endorsement post.
343    *                                      Endorsement posts are not endorsable.
344    *                                      Endorsements are not transitive. For
345    *                                      instance, if post A is endorsed by post
346    *                                      B, and an account wants to endorse B, in
347    *                                      fact, the endorsement must refers to A.
348    */
349   @Override
350   public int endorsePost(String handle, int id) throws HandleNotRecognisedException,
          PostIDNotRecognisedException, NotActionablePostException {
351         String formattedMessage;
352
353         if (Account.doesHandleExist(handle) == false) {
354            throw new HandleNotRecognisedException("the handle is not recognised");
355         } else if (Post.doesPostIdExist(id) == false) {
```

```
356             throw new PostIDNotRecognisedException("The post id does not exist");
357         } else if (Post.isAnEndorsement(id)){
358             throw new NotActionablePostException("The id entered refers to an endorsement");
359         }
360 
361         Endorsement newPost = new Endorsement(handle, id); //the endorsement object is created
362         for (int i = 0; i < (Post.postArrayList).size(); i++) {
363             if ((((Post.postArrayList).get(i)).getPostId()) == id){
364                 formattedMessage = String.format("EP@ %s :
                        %s",(Post.postArrayList).get(i).getAccountHandle(),
                        (Post.postArrayList).get(i).getBody()); //the formatted input is created to match the
                        template
365                 newPost.setEndorsementMessage(formattedMessage);
366                 (Post.postArrayList).add(newPost); //the new endorsement is added to the total array of
                        posts
367                 ((Post.postArrayList).get(i)).setEndorsements(((Post.postArrayList).get(i)).getEndorsementNumber());//
                        to the post being endorsed, we add one to its count. So it is aware of the endorsements
                        occuring
368                 return newPost.getPostId();
369             }
370 
371         }
372         return 0; //this return zero is here as the sole return cannot exist in an if statement
373     }
374 
375 
376 
377 
378 
379 
380 
381 /**
382  * The method creates a comment post referring to an existing post, similarly to
383  * a reply on Twitter. A comment post is a special post. It contains a reference
384  * to the post being commented upon.
385  * <p>
386  * The state of this SocialMediaPlatform must be be unchanged if any exceptions
387  * are thrown.
388  *
389  * @param handle of the account commenting a post.
390  * @param id    of the post being commented.
391  * @param message the comment post message.
392  * @return the sequential ID of the created post.
393  * @throws HandleNotRecognisedException if the handle does not match to any
394  *                                      account in the system.
395  * @throws PostIDNotRecognisedException if the ID does not match to any post in
396  *                                      the system.
397  * @throws NotActionablePostException if the ID refers to a endorsement post.
398  *                                      Endorsement posts are not endorsable.
399  *                                      Endorsements cannot be commented. For
400  *                                      instance, if post A is endorsed by post
401  *                                      B, and an account wants to comment B, in
402  *                                      fact, the comment must refers to A.
403  * @throws InvalidPostException     if the comment message is empty or has
404  *                                      more than 100 characters.
```

```
405      */
406
407     @Override
408     public int commentPost(String handle, int id, String message) throws HandleNotRecognisedException,
409           PostIDNotRecognisedException, NotActionablePostException, InvalidPostException {
410
411              if (Account.doesHandleExist(handle) == false) {
412                 throw new HandleNotRecognisedException("the handle does not match any in the system");
413              } else if (Post.doesPostIdExist(id) == false) {
414                 throw new PostIDNotRecognisedException("the id does not exist in the system");
415              } else if (Post.isAnEndorsement(id) == true) {
416                 throw new NotActionablePostException("The id refers to an endorsement post");
417              } else if (Post.isPostInvalid(message) == true) {
418                 throw new InvalidPostException("Either the comment is empty or it is more than 100
                       characters");
419              }
420
421              Post newComment = new Comment(handle,id,message);// no longer upcasting
422              (Post.postArrayList).add(newComment);
423              for (int i = 0; i < (Post.postArrayList).size(); i++) {
424                 if ((((Post.postArrayList).get(i)).getPostId()) == id){
425                    ((Post.postArrayList).get(i)).setNumberOfComments(((Post.postArrayList).get(i)).getCommentNUmber());
                          //one is incremenented to the commented posts post tally
426
427              return newComment.getPostId();
428
429              }
430           }
431           return 0; // the if statement cannot house the only return statement
432        }
433
434     /**
435      * The method removes the post from the platform. When a post is removed, all
436      * its endorsements should be removed as well. All replies to this post should
437      * be updated by replacing the reference to this post by a generic empty post.
438      * <p>
439      * The generic empty post message should be "The original content was removed
440      * from the system and is no longer available.". This empty post is just a
441      * replacement placeholder for the post which a reply refers to. Empty posts
442      * should not be linked to any account and cannot be acted upon, i.e., it cannot
443      * be available for endorsements or replies.
444      * <p>
445      * The state of this SocialMediaPlatform must be be unchanged if any exceptions
446      * are thrown.
447      *
448      * @param id ID of post to be removed.
449      * @throws PostIDNotRecognisedException if the ID does not match to any post in
450      *                                      the system.
451      */
452     @Override
453     public void deletePost(int id) throws PostIDNotRecognisedException {
454
455
456        if (Post.doesPostIdExist(id) == false) {
457           throw new PostIDNotRecognisedException("the post with this id does not exist");
```

```
458         }
459
460
461         for (int i = 0; i < Post.postArrayList.size(); i++) {
462             // delete endorsement posts. Since there are no comments, there's no need to point to a generic
                    empty post
463             if (((Post.postArrayList).get(i) instanceof Endorsement) &&
                        (((Post.postArrayList).get(i)).getPostId()) == id) {
464
465                 // remove one of the number of endorsements here
466                 for (int j = 0; j < Post.postArrayList.size(); j++) {
467                 if ((((Post.postArrayList).get(i)).getOriginalPostId()) ==
                            (((Post.postArrayList).get(j)).getPostId())) {
468                 (Post.postArrayList.get(j)).numberOfEndorsements =
                            (Post.postArrayList.get(j)).getEndorsementNumber() - 1 ; } // remove the log of the
                            endorsement
469             } (Post.postArrayList).remove(i);
470
471         }
472             else if (((Post.postArrayList).get(i) instanceof Comment) &&
                        ((((Post.postArrayList).get(i)).getPostId()) == id)) {
473                 // remove one of the number of comments here
474                 for (int j = 0; j < (Post.postArrayList).size(); j++) {
475                 if ((((Post.postArrayList).get(i)).getOriginalPostId()) ==
                            (((Post.postArrayList).get(j)).getPostId())) {
476                 ((Post.postArrayList).get(j)).numberOfComments = (Post.postArrayList.get(j)).getCommentNUmber()
                            - 1 ; } // remove the log of the endorsement
477
478
479                 }if (Post.doesItHaveChildrenPost(id) == false){
480                    (Post.postArrayList).remove(i);
481             }
482
483         }
484
485             // this deletes the original post, it will also work for an endorsement post
486             if ((((Post.postArrayList).get(i)).getPostId()) == id) {
487                 ((Post.postArrayList).get(i)).setBody("The original content was removed from the system and is
                            no longer available."); // the post descriptionn is changed to a generic emoty post
488                 (Post.postArrayList.get(i)).setHandle(null);// the handle is nullified
489                 (Post.postArrayList.get(i)).numberOfEndorsements = 0; // set the number of endorsements to 0
490                 //(Post.postGraveyard).add((Post.postArrayList).get(i)); // the post is then moved to a post
                            graveyard so if ever needed we can use it to link its comments we can
491                 //(Post.postArrayList).remove(i); // the post is removed from the post array list
492             }
493
494
495
496
497     } }
498
499     /**
500      * The method generates a formated string containing the details of a single
501      * post. The format is as follows:
502      *
```

```
503      * <pre>
504      * ID: [post ID]
505      * Account: [account handle]
506      * No. endorsements: [number of endorsements received by the post] | No. comments: [number of comments
             received by the post]
507      * [post message]
508      * </pre>
509      *
510      * @param id of the post to be shown.
511      * @return a formatted string containing post's details.
512      * @throws PostIDNotRecognisedException if the ID does not match to any post in
513      *                                      the system.
514      */
515     @Override
516     public String showIndividualPost(int id) throws PostIDNotRecognisedException {
517
518        if (Post.doesPostIdExist(id) == false) {
519           throw new PostIDNotRecognisedException("The id does not match any post in the system"); //If the
                  Post Id does not point to any post in the system, an exception is thrown.
520        }
521        String postOutput = "";
522        for (int k = 0 ; k < (Post.postArrayList).size(); k++){
523           if ((((Post.postArrayList).get(k)).getPostId()) == id) {
524              // the new lines are in the string in order to format the new lines in 8
525              postOutput = String.format("ID : %s \nAccount: %s \nNo. endorsements: %s | No. comments : %s\n%s
                       ",((Post.postArrayList).get(k)).getPostId(),((Post.postArrayList).get(k)).getAccountHandle(),((Post.pos
526              break;
527           }
528        }
529
530
531        return postOutput; //Returns the formatted information of the requested post.
532
533     }
534
535
536     /**
537      * The method builds a StringBuilder showing the details of the current post and
538      * all its children posts. The format is as follows (you can use tabs or spaces to represent
             indentation):
539      *
540      * <pre>
541      * {@link #showIndividualPost(int) showIndividualPost(id)}
542      * |
543      * [for reply: replies to the post sorted by ID]
544      * |  > {@link #showIndividualPost(int) showIndividualPost(reply)}
545      * </pre>
546      *
547      * See an example:
548      *
549      * <pre>
550      * ID: 1
551      * Account: user1
552      * No. endorsements: 2 | No. comments: 3
553      * I like examples.
```

```
     * |
     * | > ID: 3
     *    Account: user2
     *    No. endorsements: 0 | No. comments: 1
     *    No more than me...
     *    |
     *    | > ID: 5
     *        Account: user1
     *        No. endorsements: 0 | No. comments: 1
     *        I can prove!
     *        |
     *        | > ID: 6
     *            Account: user2
     *            No. endorsements: 0 | No. comments: 0
     *            prove it
     * | > ID: 4
     *    Account: user3
     *    No. endorsements: 4 | No. comments: 0
     *    Can't you do better than this?
     *
     * | > ID: 7
     *    Account: user5
     *    No. endorsements: 0 | No. comments: 1
     *    where is the example?
     *    |
     *    | > ID: 10
     *        Account: user1
     *        No. endorsements: 0 | No. comments: 0
     *        This is the example!
     * </pre>
     *
     * Continuing with the example, if the method is called for post ID=5
     * ({@code showIndividualPost(5)}), the return would be:
     *
     * <pre>
     * ID: 5
     * Account: user1
     * No. endorsements: 0 | No. comments: 1
     * I can prove!
     * |
     * | > ID: 6
     *    Account: user2
     *    No. endorsements: 0 | No. comments: 0
     *    prove it
     * </pre>
     *
     * @param id of the post to be shown.
     * @return a formatted StringBuilder containing the details of the post and its
     *        children.
     * @throws PostIDNotRecognisedException if the ID does not match to any post in
     *                                      the system.
     * @throws NotActionablePostException if the ID refers to an endorsement post.
     *                                    Endorsement posts do not have children
     *                                    since they are not endorsable nor
     *                                    commented.
```

```
609      */
610
611     @Override
612     public StringBuilder showPostChildrenDetails(int id)
613        throws PostIDNotRecognisedException, NotActionablePostException {
614
615        //The following assertion checks that there are at least 2 posts in the system
616        assert ((Post.postArrayList).size()) >= 2 : "There are no posts with children." ;
617
618
619        if (Post.doesPostIdExist(id) == false) {
620           throw new PostIDNotRecognisedException("the post with this id does not exist on this system");
621        } else if (Post.isAnEndorsement(id) == true){
622           throw new NotActionablePostException("The post id inputted is for that of an endorsement");
623        }
624         StringBuilder hierarchy = new StringBuilder();
625        buildObjectHierarchy(id, hierarchy, 0); //buildObjectHierarchy is called in order to buidl the string
             output.
626        return hierarchy;
627
628        }
629
630
631
632
633     /**
634      * This recursive function takes in an id, a StringBuilder object and an integer level. It then creates
           a new
635      * SocialMedia object(in order to use the showIndividualPost since it is not static method) and checks
           if the id is 0. If it is, it returns. If it isn't, it loops through
636      * the level and appends two spaces to the StringBuilder object. It then appends the post with the id
637      * to the StringBuilder object and then loops through the postArrayList and checks if the
638      * originalPostId is equal to the id. If it is, it calls the function again with the originalPostId,
639      * the StringBuilder object and the level + 1
640      *
641      * @param id The id of the post you want to start with.
642      * @param sb StringBuilder object
643      * @param level This is the level of the hierarchy. This helps determine the ammount of indentation
644      * @throws PostIDNotRecognisedException if the ID does not match to any post in
645      *                                      the system
646      */
647     protected static void buildObjectHierarchy(int id, StringBuilder sb, int level) throws
         PostIDNotRecognisedException {
648          SocialMedia newPost = new SocialMedia(); // this is created so that we can use the show individual
               post method as it is not staticr
649
650          if (id == 0){
651          return; //if the id passed is zero the recursion is ended.
652                }
653          try{
654          String[] postArr = (newPost.showIndividualPost(id)).split("\n"); // a new array is created post
               passed that is split by \n
655          sb.append("\n"); // a newline is added to give some space between posts
656          for (int j = 0; j < postArr.length;j++){
657             sb.append("\n");
```

```
658         for (int i = 0; i < level; i++) {
659         sb.append("  "); //this space is added in order to ident the posts accordingly, to give a tree
                hierachy
660         } sb.append(postArr[j]);}} catch (PostIDNotRecognisedException e){} //the post is then added to the
                stringbuildider
661
662 // the post array list is looped thrrough here and if the current post is a parent post then the method
        calls itself again. We decided using a recursive method was the best way around
663         for (Post post : Post.postArrayList) {
664            if ((post instanceof Endorsement) == false){
665               if (post.getOriginalPostId() == id){
666                  buildObjectHierarchy(post.getPostId(), sb, level + 1);
667               } }
668            }
669    }
670
671
672
673
674
675
676
677
678
679    // End Post-related methods ****************************************
680
681
682
683    // Analytics-related methods ****************************************
684
685    /**
686     * This method returns the current total number of accounts present in the
687     * platform. Note, this is NOT the total number of accounts ever created since
688     * the current total should discount deletions.
689     *
690     * @return the total number of accounts in the platform.
691     */
692    @Override
693    public int getNumberOfAccounts() {
694       return (Account.accountArrayList).size(); //Returns the number of accounts that exist in the system.
695    }
696
697    /**
698     * This method returns the current total number of original posts (i.e.,
699     * disregarding endorsements and comments) present in the platform. Note, this
700     * is NOT the total number of posts ever created since the current total should
701     * discount deletions.
702     *
703     * @return the total number of original posts in the platform.
704     */
705    @Override
706    public int getTotalOriginalPosts() {
707       int originalPostcount = 0;
708       for (int i = 0; i < (Post.postArrayList).size(); i++) {
709          if ((Post.postArrayList).get(i) instanceof Comment) { //Comments aren't original posts, so we
```

```
            disregard them here
710             continue;
711         } else if ((Post.postArrayList).get(i) instanceof Endorsement) { //Endorsements are also not
                original posts, so we disregard them here.
712             continue;
713         }else if (((Post.postArrayList).get(i)).getAccountHandle() == null) {continue;}
714         else {
715             originalPostcount +=1; //original posts aree incremented by 1
716         }
717       }
718       return originalPostcount;
719     }
720
721     /**
722      * This method returns the current total number of endorsement posts present in
723      * the platform. Note, this is NOT the total number of endorsements ever created
724      * since the current total should discount deletions.
725      *
726      * @return the total number of endorsement posts in the platform.
727      */
728     @Override
729     public int getTotalEndorsmentPosts() {
730       int endorsementCount = 0;
731       for (int i = 0; i < (Post.postArrayList).size(); i++) {
732         if (((Post.postArrayList).get(i) instanceof Endorsement) &&
                ((Post.postArrayList).get(i)).getAccountHandle() != null) { //If in an endorsement is held in
                the Post ArrayList, it has not been deleted, and so can be counted.
733             endorsementCount++ ;
734         }
735       }
736       return endorsementCount; //Returns the number of endorsements present in the system.
737   }
738
739     /**
740      * This method returns the current total number of comments posts present in the
741      * platform. Note, this is NOT the total number of comments ever created since
742      * the current total should discount deletions.
743      *
744      * @return the total number of comments posts in the platform.
745      */
746     @Override
747     public int getTotalCommentPosts() {
748       int commentCount = 0;
749       for (int i = 0; i < (Post.postArrayList).size(); i++) {
750         if (((Post.postArrayList).get(i) instanceof Comment) &&
                ((Post.postArrayList).get(i)).getAccountHandle() != null) { //If a comment is held the Post
                ArrayList, it has not been deleted, and so can be counted.
751             commentCount++ ;
752         }
753       }
754       return commentCount; //Returns the number of comments present in the system.
755     }
756
757     /**
758      * This method identifies and returns the post with the most number of
```

```
759      * endorsements, a.k.a. the most popular post.
760      *
761      * @return the ID of the most popular post.
762      */
763     @Override
764     public int getMostEndorsedPost(){
765        int index = 0;
766        try{
767        int max = (((Post.postArrayList).get(0)).getEndorsementNumber());
768        for (int i = 0; i < (Post.postArrayList).size(); i++) { //We iterate though the Post ArrayList to find
               the post with the most endorsements.
769           if ((Post.postArrayList).get(i) instanceof Endorsement) { //Endorsements cannot be endorsed, so we
                  ignore them here.
770              continue;
771           }
772           else if (((Post.postArrayList).get(i)).getEndorsementNumber() > max ) { //If we found a post with
                  the most endorsements we've seen so far in our search, it becomes the "running winner".
773              index = i;
774              max = ((Post.postArrayList).get(i)).getEndorsementNumber();
775           }
776        }
777        return ((Post.postArrayList).get(index)).getPostId(); //After we've iterated through the entire Post
               ArrayList, the "Running Winner" will be the post with the most endorsements in the entire system.
778     }catch(NullPointerException e){e.printStackTrace();}catch(IndexOutOfBoundsException
            e){e.printStackTrace();}
779        return 0; //this will never be called. It is just there to keep the compiler happy
780     }
781
782     /**
783      * This method identifies and returns the account with the most number of
784      * endorsements, a.k.a. the most popular account.
785      *
786      * @return the ID of the most popular account.
787      */
788     @Override
789     public int getMostEndorsedAccount() {
790        // a hashmap (/dictionary) is created
791        HashMap<String, Integer> endorsementLeaderboard = new HashMap<String, Integer>();
792
793     try{
794        for (int i = 0; i < (Account.accountArrayList).size(); i++){
795           endorsementLeaderboard.put(((Account.accountArrayList).get(i)).getHandle(),0); //We add all the
                  accounts in the system to our "Leaderboard" to prepare for the counting of each account's total
                  endorsements.
796        }
797
798        for (int k = 0; k < (Post.postArrayList).size(); k++) {
799           if ((((Post.postArrayList).get(k) instanceof Endorsement) ||
                  (((Post.postArrayList).get(k)).getAccountHandle() == null)) || ((Post.postArrayList).get(k)
                  instanceof Comment)){ //Endorsements cannot be endorsed, so we ignore them here.
800              continue;
801           } else {
802              endorsementLeaderboard.put(((Post.postArrayList).get(k)).getAccountHandle(),
                     (endorsementLeaderboard.get(((Post.postArrayList).get(k)).getAccountHandle()) +
                     ((Post.postArrayList).get(k)).getEndorsementNumber() ));
```

```
803             //For each non-endorsement post in the system, we add its number of endorsements to its
                    account's total in the leaderboard.
804          }
805       }
806
807       String mostPopular = "";
808       int highest = 0;
809
810       for (String j : endorsementLeaderboard.keySet()) {//We iterate through the Leaderboard to find the
              account with the highest number of endorsements.
811          if ((endorsementLeaderboard.get(j)) > highest) {//If we find an account in the leaderboard with the
                 highest number of endorsements so far in our search, it becomes the "running winner"
812             mostPopular = j;
813             highest = endorsementLeaderboard.get(j);
814             //After we've iterated through the entire Leaderboard, the "Running Winner" will be the account
                    with the highest number of endorsemnets on its collective posts.
815          }
816       }
817
818       //Loop through the account array list to get the id
819       for (int i = 0; i < (Account.accountArrayList).size(); i++){
820          if (((Account.accountArrayList).get(i)).getHandle() == mostPopular){
821             return (Account.accountArrayList).get(i).getAccountId();//We return the ID of the most popular
                    account.
822          }
823       }}catch(NullPointerException e){e.printStackTrace();}
824       catch(IndexOutOfBoundsException e){e.printStackTrace();}
825
826       return 0; // the if statement cannot house the only return statement
827
828
829
830    }
831
832    // End Analytics-related methods ****************************************
833
834    // Management-related methods ****************************************
835
836    /**
837     * Method empties this SocialMediaPlatform of its contents and resets all
838     * internal counters.
839     */
840    @Override
841    public void erasePlatform(){
842       (Account.accountArrayList).clear();
843       (Post.postArrayList).clear();
844       (Post.postGraveyard).clear();
845       Post.setNumberOfPostsToZero();
846       //All the ArrayLists will be wiped of their objects and the tracked number of posts will be set to 0.
847    }
848
849    /**
850     * Method saves this SocialMediaPlatform's contents into a serialised file, with
851     * the filename given in the argument.
852     *
```

```
853      * @param filename location of the file to be saved
854      * @throws IOException if there is a problem experienced when trying to save the
855      *                   store contents to the file
856      */
857     @Override
858     public void savePlatform(String filename) throws IOException{
859        /* all the static array in Account and Post class are moved to variables in this method to get around
860        serialising static attributes
861         *
862        */
863        ArrayList<Account> arrOfAccounts = Account.accountArrayList;
864        ArrayList<Post> arrOfPosts = Post.postArrayList;
865        ArrayList<Integer> registerOfRandomNUmbers = Account.randomNumberArray;
866        ArrayList<Post> arrPostGraveyard = Post.postGraveyard;
867
868
869        try (FileOutputStream fos = new FileOutputStream(filename); //start the file output stream
870        ObjectOutputStream oos = new ObjectOutputStream(fos); )
871        {
872           //the objects are written to the passed file, that is assumed to be empty/
873           oos.writeObject(arrOfAccounts);
874           oos.writeObject(arrOfPosts);
875           oos.writeObject(registerOfRandomNUmbers);
876           oos.writeObject(arrPostGraveyard);
877           oos.writeObject(Integer.valueOf(Post.numberOfPosts));
878        } catch (FileNotFoundException e){
879           System.out.println("Sorry the file was not found");
880        } catch (IOException e){
881           throw new IOException("There's been a problem with the input output");
882        }
883
884
885     }
886
887     /**
888      * Method should load and replace this SocialMediaPlatform's contents with the
889      * serialised contents stored in the file given in the argument.
890      * <p>
891      * The state of this SocialMediaPlatform's must be be unchanged if any
892      * exceptions are thrown.
893      *
894      * @param filename location of the file to be loaded
895      * @throws IOException         if there is a problem experienced when trying
896      *                             to load the store contents from the file
897      * @throws ClassNotFoundException if required class files cannot be found when
898      *                             loading
899      */
900     @Override
901     public void loadPlatform(String filename) throws IOException, ClassNotFoundException{
902        //an attempt at deserialisation
903        // the input stream is initialised
904        try(FileInputStream fis = new FileInputStream(filename);
905        ObjectInputStream ois = new ObjectInputStream(fis);){
906           for (int i = 0; i < 5; i++){ //a fixed loop is created to go through each of the objects
907              Object obj = ois.readObject(); // the object is read
```

```
908             if (obj instanceof ArrayList){
909                switch (i){ //a switch is used because we thought it'd be more efficient than if statements
910                   case 0:
911                      Account.accountArrayList = (ArrayList<Account>) obj; //an attempt at safely casting
912                      break;
913                   case 1:
914                      Post.postArrayList = (ArrayList<Post>) obj;
915                      break;
916                   case 2:
917                      Account.randomNumberArray = (ArrayList<Integer>) obj;
918                      break;
919
920                   case 3:
921                      Post.postGraveyard = (ArrayList<Post>) obj;
922                      break;
923
924
925                   }
926                }else if (obj instanceof Integer){
927                   Post.numberOfPosts = (Integer) obj;
928                }
929
930
931
932          }
933
934       }
935
936
937    }
938
939    // End Management-related methods ***************************************
940
941 }
```

2 Account.java

```
1  // A package declaration. It is used to group classes together.
2  package socialmedia;
3  import java.util.ArrayList;
4  import java.util.Random;
5  import java.io.Serializable;
6
7
8  /**
9   * The Account class is a class that stores the account's ID, handle and description. It also has
10  * static methods that generate a unique random number, check if the account ID is recognised, check if
11  * the handle exists, check if the handle contains white space or is empty, check if the handle is
12  * invalid, and getter and setter methods.
13  */
14 public class Account implements Serializable {
15
16
17     // Instance Attributes
```

```
18      public int accountId; //an account's id
19      public String handle; // the handle of the account
20      public String description; // the account's description
21
22
23      //Static Attribute - An ArrayList to store the system's Accounts
24      public static ArrayList<Account> accountArrayList = new ArrayList<Account>();
25      public static ArrayList<Integer> randomNumberArray = new ArrayList<Integer>(); // This is a static
            attribute that stores the the random numbers generated
26
27
28      //Static Methods
29
30      /**
31       * It generates a random number (no parameters are passed) and checks to see if it's already in the
             array of previously generated numbers,
32       * and if it is, it generates another random number.
33       * While the finla line says it returns zero, it was likely never be used.
34       *
35       * @return The return value is an integer.
36       */
37      public static int generateUniqueRandomNumber() {
38           //this bit for if this is the first run
39           Random r = new Random( System.currentTimeMillis() );
40           int randomNumber = ((1 + r.nextInt(9)) * 10000 + r.nextInt(10000));
41           if (randomNumberArray.contains(randomNumber) == false){
42             randomNumberArray.add(randomNumber);
43             return randomNumber;}
44             else{return generateUniqueRandomNumber();}
45             //make an assertion later on that the generate return number is not a zero
46      }
47
48
49      /**
50       * This method returns a boolean, depending on whether the account id is in the system or not.
51       * True if the account ID exists, and false otherwise.
52       *
53       * @param Id The account ID to be checked
54       * @return The method isAccountIdRecognised is returning a boolean value.
55       */
56      public static boolean isAccountIdRecognised(int Id) {
57        for (int i = 0 ; i < accountArrayList.size(); i++) {
58          if (((accountArrayList.get(i)).getAccountId()) == Id) {
59            return true;
60          }
61        }
62        return false;
63      }
64
65      /**
66       * This function takes in a string and returns a boolean. It returns true if the string is equal to
67       * the handle of any account in the accountArrayList. It returns false if the string is not equal
68       * to the handle of any account in the accountArrayList
69       *
70       * @param handle the handle of the account to be checked
```

```
71       * @return The method is returning a boolean value.
72       */
73      public static boolean doesHandleExist(String handle) {
74        for (int i = 0; i < accountArrayList.size(); i++){
75          if (((accountArrayList.get(i)).getHandle()).equals(handle)){
76            return true;
77          }
78        }
79        return false;
80      }
81
82
83      /**
84       * It returns true if the string contains a space or is empty
85       *
86       * @param handle The handle to check for white space or emptiness.
87       * @return The method is returning a boolean value.
88       */
89      public static boolean doesItContainWhiteSpaceOrIsEmpty(String handle) {
90        for (int i = 0; i < handle.length(); i++) {
91          //this loops through the handle to see if it has whitespace or it's empty
92          if ((handle.charAt(i)) == (' ')) {
93            return true;
94          }
95        }
96        if (handle.equals("")) {
97          return true;
98        } else{return false;}
99      }
100
101
102     /**
103      * If the handle contains white space, is empty, or if the handle is longer than 30 characters,
104      * the method returns true. Otherwise, it returns false.
105      *
106      * @param handle The handle to be checked
107      * @return The method is returning a boolean value.
108      */
109     public static boolean isHandleInvalid(String handle) {
110       if (doesItContainWhiteSpaceOrIsEmpty(handle) == true || handle.length() > 30) {
111         return true;
112       } else {return false;}
113     }
114
115
116     //Getter Methods
117     /**
118      * This is a getter method that returns the accountId of the account
119      *
120      * @return The accountId
121      */
122     public int getAccountId() {
123       return accountId;
124     }
125
```

```
126     /**
127      * This function returns the handle of the user
128      *
129      * @return The handle of the user.
130      */
131     public String getHandle() {
132       return handle;
133     }
134 
135     /**
136      * This function returns the description of the object
137      *
138      * @return The description of the item.
139      */
140     public String getDescription() {
141       return description;
142     }
143 
144 
145 
146     //Setter Methods
147     /**
148      * This function sets the accountId to the newAccountId
149      *
150      * @param newAccountId The new account ID to set.
151      */
152     public void setAccountId(int newAccountId) {
153       this.accountId = newAccountId;
154     }
155 
156     /**
157      * This function sets the handle of the user to the new handle
158      *
159      * @param newHandle The new handle to set.
160      */
161     public void setHandle(String newHandle) {
162       this.handle = newHandle;
163     }
164 
165     /**
166      * This function sets the description of the object to the newDescription parameter
167      *
168      * @param newDescription The new description of the item.
169      */
170     public void setDescription(String newDescription) {
171       this.description = newDescription;
172     }
173 
174 }
```

3 Post.java

```
1  package socialmedia;
2 
```

```java
import java.util.ArrayList;
import java.io.*;

/**
 * This class is used to create a post object
 */
public class Post implements Serializable{


  //Instance Attributes
  public int postId = ++numberOfPosts;
  public String body;
  public String handle; // the account handle a post is linked to
  public int numberOfEndorsements; //keeps track of the number of endorsements a post has
  public int numberOfComments; //keeps track of the number of comments a post has
  public int originalPostID;// gives the parent post id (if it has one)



  /**
   * This function takes a string and returns true if the string is empty or longer than 100 characters
   *
   * @param message The message that the user is trying to post.
   * @return A boolean value.
   */
  public static boolean isPostInvalid(String message) {
    if (message == "") {
      return true;
    } else if (message.length() > 100) {
      return true;
    } else {return false;}
  }
  /**
   * It checks if the post has any children posts.
   *
   * @param id the id of the post
   * @return A boolean value.
   */
  public static boolean doesItHaveChildrenPost(int id){
    for (int i = 0; i < postArrayList.size(); i++) {
      if (((postArrayList.get(i)).getOriginalPostId() == id) && (postArrayList.get(i).getAccountHandle() ==
          null)) {
        return true;
      }
    }
    return false;
  }


  /**
   * It loops through the array list of posts and checks if the post id of the post at the current
   * index is equal to the id passed in as a parameter. If it is, it returns true. If it isn't, it
   * returns false
   *
   * @param id the id of the post
```

```
57     * @return A boolean value
58     */
59    public static boolean doesPostIdExist(int id) {
60      for (int i = 0; i < postArrayList.size(); i++) {
61        if (((postArrayList.get(i)).getPostId() == id) || (postArrayList.get(i).getAccountHandle() == null)) {
62          return true;
63        }
64      }
65      return false;
66    }
67
68
69    /**
70     * This function checks if a post is an endorsement by checking if the post is an instance of the
71     * Endorsement class
72     *
73     * @param id the id of the post
74     * @return The method is returning a boolean value.
75     */
76    public static boolean isAnEndorsement(int id) {
77      for (int i = 0; i < postArrayList.size(); i++) {
78        if (((postArrayList.get(i)).getPostId()) == id) {
79          if (postArrayList.get(i) instanceof Endorsement) {
80            return true;
81          }
82        }
83      }
84      return false;
85    }
86
87
88
89
90    //Static Attribute - An ArrayList to store the system's Posts
91    public static ArrayList<Post> postArrayList = new ArrayList<Post>();
92    public static ArrayList<Post> postGraveyard = new ArrayList<Post>();
93    public static int numberOfPosts = 0;
94
95
96    //Getter methods
97    /**
98     * This function returns the postId of the post
99     *
100    * @return The postId
101    */
102   public int getPostId(){
103     return postId;
104   }
105   /**
106    * This function returns the account handle of the user
107    *
108    * @return The handle of the account.
109    */
110   public String getAccountHandle(){
111     return handle;
```

```
112   }
113   /**
114    * // Java
115    * public String getBody(){
116    *     return body;
117    *   }
118    *
119    * @return The body of the email.
120    */
121   public String getBody(){
122     return body;
123   }
124   /**
125    * This function returns the number of comments on a post
126    *
127    * @return The number of comments.
128    */
129   public int getCommentNUmber(){
130     return numberOfComments;
131   }
132   /**
133    * This method returns the number of endorsements a post has
134    *
135    * @return The number of endorsements.
136    */
137   public int getEndorsementNumber(){
138     return numberOfEndorsements;
139   }
140   /**
141    * This function returns the original post ID of the post (Posts being Endorsements and Comments).
142    * If Post id is zero, then it is an original post.
143    *
144    * @return The original post ID.
145    */
146   public int getOriginalPostId(){
147     return originalPostID;
148   }
149
150   /**
151    * This function takes in a string parameter and returns an integer. The integer is the total number
152    * of posts that the user has made
153    *
154    * @param handle the account handle of the account you want to get the post count of
155    * @return The total number of posts made by a user.
156    */
157   public static int getTotalPostCount(String handle){
158     int count = 0;
159     for (int k = 0; k < (Post.postArrayList).size(); k++) {
160       if (Post.postArrayList.get(k).getAccountHandle() == handle){
161         count++;
162       } else{continue;}
163
164   }
165   return count;
166 }
```

```
167   /**
168    * This function takes in a string, and returns an integer. The integer is the total number of
169    * endorsements for all posts made by the account with the handle that was passed in
170    *
171    * @param handle the account handle of the account you want to get the endorsement total of
172    * @return The total number of endorsements for a given account.
173    */
174   public static int getAccountEndorsementTotal(String handle){
175     int count = 0;
176     for (int k = 0; k < (Post.postArrayList).size(); k++) {
177          if ((Post.postArrayList).get(k) instanceof Endorsement) {
178             continue;
179          } else if (Post.postArrayList.get(k).getAccountHandle() == handle){
180         count += Post.postArrayList.get(k).getEndorsementNumber();
181         }
182       }
183     return count;
184   }
185
186   //Seter methods
187   /**
188    * This function sets the postId to the newPostId passed.
189    *
190    * @param newPostId The new post ID to set.
191    */
192   public void setPostId(int newPostId) {
193     this.postId = newPostId;
194   }
195   /**
196    * // Java
197    * public static void setNumberOfPostsToZero(){
198    *     numberOfPosts = 0;
199    *   }
200    */
201   public static void setNumberOfPostsToZero(){
202     numberOfPosts = 0;
203   }
204   /**
205    * This function sets the handle of the user
206    *
207    * @param handle The handle of the user to be followed.
208    */
209   public void setHandle(String handle) {
210     this.handle = handle;
211   }
212   /**
213    * The function takes in an integer, increments it by one, and then sets the value of the variable
214    * numberOfEndorsements to the new value
215    *
216    * @param endorsementNumber The number of endorsements the user has.
217    */
218   public void setEndorsements(int endorsementNumber){
219     numberOfEndorsements = ++endorsementNumber;
220   }
221   /**
```

```
222     * This function takes an integer as a parameter and sets the numberOfComments variable to the value
223     * of the parameter plus one
224     *
225     * @param commentNumber The number of comments that the post has.
226     */
227    public void setNumberOfComments(int commentNumber){
228      numberOfComments = ++commentNumber;
229    }
230
231    /**
232     * This function sets the body of the post
233     *
234     * @param body The body of the message.
235     */
236    public void setBody(String body) {
237      this.body = body;
238    }
239
240
241    //Constructor
242    /** Default constructor for post object
243     */
244    public Post(){
245
246    }
247    // This constructor creates a post object
248    public Post(String handle,String body){
249      this.body = body;
250      this.handle = handle;
251
252    }
253
254
255
256
257  }
258
259    /**
260     * The Endorsement class is a subclass of the Post class. It has a constructor that takes in a handle
261     * and an id. It also has a method called setEndorsementMessage that takes in a message and sets the
262     * endorsementMessage variable to that message
263     */
264    class Endorsement extends Post{
265
266    public String endorsementMessage;
267
268  // This constructor creates an endorsement object
269    public Endorsement(String handle,int id){
270      super();
271      this.handle = handle;
272      this.originalPostID = id;
273    }
274
275    /**
276     * This function sets the endorsement message to the message passed in as a parameter
```

```
277     *
278     * @param message The endorsement message to be displayed on the endorsement.
279     */
280    public void setEndorsementMessage(String message){
281      endorsementMessage = message;
282    }
283    }
284
285   /**
286    * A subclass Comment is created, which extends the Post superclass.
287    */
288   class Comment extends Post{
289
290
291  // Comment Constructor
292  public Comment(String handle, int id, String message){
293    super();
294    this.handle = handle;
295    body = message;
296    originalPostID = id;
297  }
298  }
```