SocialMedia.java

```java
package socialmedia;

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


public class SocialMedia implements SocialMediaPlatform {
    private ArrayList<Account> accountList = new ArrayList<Account>();
    private ArrayList<Post> emptyPostList = new ArrayList<Post>();

    @Override
    public int createAccount(String handle) throws IllegalHandleException,
InvalidHandleException {
        Account.validateHandle(handle, accountList);
        Account newAccount = new Account(handle);
        int numOfAccounts = getNumberOfAccounts();
        accountList.add(newAccount);
        assert (numOfAccounts + 1 == getNumberOfAccounts()) : "Number of
accounts has not increased.";
        return newAccount.getID();
    }

    @Override
    public int createAccount(String handle, String description) throws
IllegalHandleException, InvalidHandleException {
        Account.validateHandle(handle, accountList);
        Account newAccount = new Account(handle, description);
        int numOfAccounts = getNumberOfAccounts();
        accountList.add(newAccount);
        assert (numOfAccounts + 1 == getNumberOfAccounts()) : "Number of
accounts has not increased.";
        return newAccount.getID();
    }

    @Override
    public void removeAccount(int id) throws AccountIDNotRecognisedException {
        Account accountToDelete = Account.findAccountById(id, accountList);
        int numOfAccounts = getNumberOfAccounts();
        while (accountToDelete.getPosts().size()>0){ // Deletes all posts from
the account
            try{
                Post p = accountToDelete.getPosts().get(0);
                deletePost(p.getID());
            } catch(PostIDNotRecognisedException e){
```

```
                continue;
            }
        }
        accountList.remove(accountToDelete);
        assert (numOfAccounts - 1 == getNumberOfAccounts()) : "Number of
accounts has not decreased.";
    }

    @Override
    public void removeAccount(String handle) throws
HandleNotRecognisedException {
        Account accountToDelete = Account.findAccountByHandle(handle,
accountList);
        try{
            removeAccount(accountToDelete.getID());
        } catch (AccountIDNotRecognisedException e){

        }
    }

    @Override
    public void changeAccountHandle(String oldHandle, String newHandle)
            throws HandleNotRecognisedException, IllegalHandleException,
InvalidHandleException {
        Account.validateHandle(newHandle, accountList);
        Account account = Account.findAccountByHandle(oldHandle, accountList);
        account.setHandle(newHandle);
        assert (account.getHandle() == newHandle) : "Handle has not updated.";

    }

    @Override
    public void updateAccountDescription(String handle, String description)
throws HandleNotRecognisedException {
        Account account = Account.findAccountByHandle(handle, accountList);
        account.setDescription(description);
        assert (account.getDescription() == description):"Description has not
updated.";
    }

    @Override
    public String showAccount(String handle) throws
HandleNotRecognisedException {
        Account account = Account.findAccountByHandle(handle, accountList);
        return account.toString();
    }

    @Override
```

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

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

```
188         private StringBuilder DFSChildren(Post originalPost, Post post, int
189     indent, StringBuilder sb, ArrayList<Post> visited) throws
190     PostIDNotRecognisedException{
191             visited.add(post); // List of posts that have been visited by search
192
193             if(post instanceof Comment){
194                 boolean isFirstComment;
195                 if (((Comment)
196     post).getReferencePost().getComments().get(0).equals(post)){ //Checks whether
197     post is the first comment under a post
198                     isFirstComment = true;
199                 } else{
200                     isFirstComment = false;
201                 }
202                 sb.append(Post.formatMessage(showIndividualPost(post.getID()),
203     indent, isFirstComment));
204             }
205             else{ //Post is an original post
206                 sb.append(showIndividualPost(post.getID())+"\n");
207             }
208             indent++;
209
210             for(Comment c:post.getComments()){ //Recursively calls on all comments
211     under the current post
212                 if(!visited.contains(c)){
213                     DFSChildren(originalPost, c, indent, sb, visited);
214                 }
215             }
216             return sb;
217         }
218
219         @Override
220         public StringBuilder showPostChildrenDetails(int id)
221                 throws PostIDNotRecognisedException, NotActionablePostException {
222             StringBuilder sb = new StringBuilder();
223             Post postToShow = Post.findPostByID(id, accountList, emptyPostList);
224
225             if (postToShow instanceof EndorsementPost){ //Cannot call method on
226     endorsement posts
227                 throw new NotActionablePostException();
228             }
229             return DFSChildren(postToShow, postToShow, 0,sb,new
230     ArrayList<Post>()); //recursively visits all comments under original post
231         }
232
233         @Override
234         public int getNumberOfAccounts() {
235             return accountList.size();
```

```
    }

    @Override
    public int getTotalOriginalPosts() {
        int total = 0;
        for(Account a:accountList){
            total += a.getOriginalPostCount();
        }
        return total;
    }

    @Override
    public int getTotalEndorsmentPosts() {
        int total = 0;
        for(Account a:accountList){
            total += a.getEndorsementCount();
        }
        return total;
    }

    @Override
    public int getTotalCommentPosts() {
        int total = 0;
        for(Account a:accountList){
            for(Post p:a.getPosts()){
                total += p.getComments().size();
            }
        }
        return total;
    }

    @Override
    public int getMostEndorsedPost() {
        Post mostEndorsedPost = null;
        for(Account a:accountList){
            for(Post p : a.getPosts()){
                if(mostEndorsedPost == null || p.getNumEndorsements() >
mostEndorsedPost.getNumEndorsements()){
                    mostEndorsedPost = p;
                }
            }
        }
        if (mostEndorsedPost == null){ // If there are no posts on the
platform
            return 0;
        }
        return mostEndorsedPost.getID();
    }
```

```
284
285         @Override
286         public int getMostEndorsedAccount() {
287             Account mostEndorsedAccount = null;
288             for(Account a : accountList){
289                 if(mostEndorsedAccount == null || a.getEndorsementCount() >
290     mostEndorsedAccount.getEndorsementCount()){
291                     mostEndorsedAccount = a;
292                 }
293             }
294             if (mostEndorsedAccount == null){ // If there are no accounts on the
295     platform
296                 return 0;
297             }
298             return mostEndorsedAccount.getID();
299         }
300
301         @Override
302         public void erasePlatform() {
303             accountList.clear();
304             emptyPostList.clear();
305             Post.resetIdCount();
306             Account.resetIdCount();
307             assert (accountList.size() == 0) : "Account list not empty";
308         }
309         @Override
310         public void savePlatform(String filename) throws IOException {
311             try(ObjectOutputStream out = new ObjectOutputStream(new
312     FileOutputStream(filename))){
313                 Account[] accountArr = accountList.toArray(new
314     Account[accountList.size()]); //Create an array of accountList
315                 out.writeObject(accountArr);
316                 Post[] emptyPostArr = emptyPostList.toArray(new
317     Post[emptyPostList.size()]); //Create an array of emptyPostList
318                 out.writeObject(emptyPostArr);
319
320                 int nextAccountID = Account.getNextId(); // Saves account ID
321     counters' value
322                 out.writeObject(nextAccountID);
323                 int nextPostID = Post.getNextId(); // Saves post ID counters'
324     value
325                 out.writeObject(nextPostID);
326             }
327
328             File f = new File(filename);
329             assert (f.isFile()) : "File has not been created";
330         }
331
```

```
    @Override
    public void loadPlatform(String filename) throws IOException,
ClassNotFoundException {
        try(ObjectInputStream in = new ObjectInputStream(new
FileInputStream(filename))){
            Account[] accountArr = (Account[])in.readObject(); // Read as
Array
            accountList = new ArrayList<>(Arrays.asList(accountArr)); // Cast
to ArrayList

            Post[] emptyPostArr = (Post[])in.readObject();
            emptyPostList = new ArrayList<>(Arrays.asList(emptyPostArr));

            int nextAccountID = (int)in.readObject();
            Account.setNextId(nextAccountID);

            int nextPostID = (int)in.readObject();
            Post.setNextId(nextPostID);


        }
    }
}
```

Account.java

```java
package socialmedia;

import java.util.ArrayList;

import java.io.Serializable;

class Account implements Serializable{
    private static int nextID = 1; //ID assigned to next account to be created
    private String handle;
    private String description;
    private int accountID;
    private int postCount = 0; // Number of original posts posted by the ac-
count
    private int endorsementCount = 0; //Number of endorsements refering to a
post posted by this account
    private boolean postCountUpToDate = false;
    private boolean endorsementCountUpToDate = false;
    private final static int MAX_HANDLE_LENGTH = 30;

    private ArrayList<Post> accountPosts = new ArrayList<Post>();//All posts
posted by the account (original, comments and endorsements)

    /**
     * Creates an instance of an Account object and calls the overloaded con-
structor with the handle and an empty description.
     * @param handle The handle of the account to be created.
     */
    public Account(String handle){
        this(handle, "");

    }
    /**
     * Creates an instance of an account object by assigning the handle and
description to the respective parameters. Also, the accountID is assigned
     * to the next  sequential ID.
     * @param handle The handle to be associated with the account.
     * @param description The description to be associated with the account.
     */
    public Account(String handle, String description){
        this.handle = handle;
        this.description = description;
        accountID = nextID++;
    }
    /**
     * Returns the handle associated with the account.
     * @return The handle of the account.
```

```
47          */
48         public String getHandle(){
49             return handle;
50         }
51         /**
52          * Returns the decscription associated with the account.
53          * @return The description of the account.
54          */
55         public String getDescription(){
56             return description;
57         }
58         /**
59          * Returns the ID associated with the account.
60          * @return The ID related with the account.
61          */
62         public int getID(){
63             return accountID;
64         }
65         /**
66          * Returns the number of original posts (excluding comments and endorse-
67     ments) the account has made.
68          * @return The number of original posts on the account.
69          */
70         public int getOriginalPostCount(){
71             if(!postCountUpToDate){
72                 int total=0;
73                 for(Post p:accountPosts){
74                     if(!(p instanceof Comment || p instanceof EndorsementPost)){
75                         total += 1;
76                     }
77                 }
78                 postCount = total; // cache the calculated value
79                 postCountUpToDate = true;
80             }
81             return postCount;
82         }
83         /**
84          * Returns the number of endorsements for posts that are made for the ac-
85     count.
86          * @return The total number of posts that are endorsing the posts made by
87     the account.
88          */
89         public int getEndorsementCount(){
90             if(!endorsementCountUpToDate){
91                 int total = 0;
92                 for(Post p:accountPosts){
93                     total += p.getNumEndorsements();
94                 }
```

```
95              endorsementCount = total; // Cache the calculated value
96              endorsementCountUpToDate = true;
97          }
98          return endorsementCount;
99      }
100     /**
101      * Returns the number of posts made by the account. This includes original
102 posts, comments and endorsements
103      * @return The number of posts made by the account.
104      */
105     public int getTotalPostCount(){
106         return accountPosts.size();
107     }
108     /**
109      * Returns the ArrayList accountPosts which contains all of the posts cre-
110 ated by the account. This includes
111      * original posts, comments and endorsements.
112      * @return ArrayList of posts created by the account
113      */
114     public ArrayList<Post> getPosts(){
115         return accountPosts;
116     }
117     /**
118      * Returns the next sequential ID that will be assigned to the next ac-
119 count to be created
120      * @return ID to be used by the next account to be created
121      */
122     public static int getNextId(){
123         return nextID;
124     }
125
126     //Setters
127     /**
128      * Sets the next sequential ID, that will be assinged to the next account
129 to be created, to the value
130      * passed in
131      * @param id The value to assign to nextID
132      */
133     public static void setNextId(int id){
134         nextID=id;
135     }
136     /**
137      * Sets the handle of the account to a new value.
138      * @param newHandle The value to change the handle of the account to.
139      */
140     public void setHandle(String newHandle){
141         handle = newHandle;
142     }
```

```
143         /**
144          * Sets the description of the account to a new value.
145          * @param newDescription The value to change the description of the ac-
146     count to.
147          */
148         public void setDescription(String newDescription){
149             description = newDescription;
150         }
151         /**
152          * Resets the sequential ID to a value of 1.
153          */
154         public static void resetIdCount(){
155             nextID = 1;
156         }
157         /**
158          * Set the endorsementCountUpToDate attribute to false, so when the
159     getEndorsementCount() method is called the number of endorsements related to
160          * the account is recalculated.
161          */
162         public void setEndorsementCountUpToDateToFalse(){
163             this.endorsementCountUpToDate = false;
164         }
165         /**
166          * Sets the boolean value of postCountUpToDate to false. postCountUpToDate
167     states whether the count of original posts
168          * is up to date.
169          */
170         public void setPostCountUpToDateToFalse(){
171             this.postCountUpToDate = false;
172         }
173         /**
174          * Adds a post to the list of posts that the account has made.
175          * @param p The post to be added to the list of posts that the account has
176     made.
177          */
178         public void addPost(Post p){
179             this.accountPosts.add(p);
180             if (!(p instanceof Comment || p instanceof EndorsementPost)){ //If
181     post is original post
182                 postCountUpToDate = false;
183             }
184 
185         }
186         /**
187          * Removes the post (specified by parameter p) from the list of posts that
188     the account has made.
189          * @param p The post to be removed from the list of posts that the account
190     has made.
```

```
191         */
192         public void removePost(Post p){
193             this.accountPosts.remove(p);
194             if (!(p instanceof Comment || p instanceof EndorsementPost)){ // If
195     post is original post
196                 postCountUpToDate = false;
197             }
198         }
199 
200         /**
201          * Returns a string that contains information about an account including
202     its unique account ID, the handle
203          * of the account, the description of the account, the number of posts the
204     account has created (original posts,
205          * comments and endorsements) and the number of endorsement posts that re-
206     fer to a post created by this account
207          *
208          * <pre>
209          * ID: 1
210          * Handle: user1
211          * Description: This is my description
212          * Post count: 1
213          * Endorse count: 1
214          * </pre>
215          */
216         @Override
217         public String toString(){
218             return  "ID: " + accountID +
219                     "\nHandle: " + handle +
220                     "\nDescription: " + description +
221                     "\nPost count: " + getTotalPostCount() +
222                     "\nEndorse count: " + getEndorsementCount();
223         }
224         /**
225          * Validates a string to be used as the handle of an account by checking
226     that it is less than 30 characters, not empty, contains
227          * no white space and is not being used by any other account in the sys-
228     tem.
229          * @param handle The string to be used as the handle of an account.
230          * @param accountList An ArrayList of account objects.
231          * @throws InvalidHandleException This is thrown when the handle is more
232     than 30 characters, or is an empty string, or contains any whitespace.
233          * @throws IllegalHandleException This is thrown when an account already
234     contains the handle defined in the parameter handle.
235          */
236         public static void validateHandle(String handle, ArrayList<Account> ac-
237     countList) throws InvalidHandleException, IllegalHandleException{
```

```
238         if(handle.length() > MAX_HANDLE_LENGTH || handle.isEmpty() || han-
239 dle.matches("(\\s)+")){ //Checks string is not empty, less than 30 characters
240 and contains no whitespace
241             throw new InvalidHandleException();
242         }
243         for(Account a : accountList){ // Checks if any account already owns
244 this handle
245             if(a.getHandle().equals(handle)){
246                 throw new IllegalHandleException();
247             }
248         }
249     }
250     /**
251      * Returns the account with the handle defined by parameter handle if it
252 exists in the ArrayList accounts, otherwise a HandleNotRecognisedException is
253 thrown.
254      * @param handle The account with the handle to find.
255      * @param accounts An ArrayList of Account objects.
256      * @return The account object that has the handle that is equal to the pa-
257 rameter handle.
258      * @throws HandleNotRecognisedException This is thrown if no account has a
259 handle defined by the parameter handle in the ArrayList of accounts (parameter
260 accounts).
261      */
262     public static Account findAccountByHandle(String handle, ArrayList<Ac-
263 count> accounts) throws HandleNotRecognisedException{
264         for (Account a : accounts){
265             if (a.getHandle().equals(handle)){
266                 return a;
267             }
268         }
269         throw new HandleNotRecognisedException();
270     }
271     /**
272      * Returns the account with the ID defined by the parameter id if it con-
273 tained in the ArrayList of accounts, otherwise an AccountIDNotRecognisedExcep-
274 tion is thrown.
275      * @param id The ID of the account to find.
276      * @param accounts An ArrayList of Accounts.
277      * @return The account with the ID that is equal to the parameter id.
278      * @throws AccountIDNotRecognisedException This is thrown if no account in
279 the ArrayList of accounts that has an ID that is equal to the parameter id.
280      */
281     public static Account findAccountById(int id, ArrayList<Account> accounts)
282 throws AccountIDNotRecognisedException{
283         for (Account a : accounts){
284             if (a.getID() == id){
285                 return a;
```

```
286                 }
287             }
288             throw new AccountIDNotRecognisedException();
289         }
290 
291     }
```

Post.java

```java
package socialmedia;

import  java.util.ArrayList;

import java.io.Serializable;

public class Post implements Serializable{
    protected static int nextID = 1;
    protected Account account;
    protected int postID;
    protected String message;
    protected ArrayList<Comment> comments = new ArrayList<Comment>();
    protected ArrayList<EndorsementPost> endorsements = new ArrayList<Endorse-
mentPost>(); //posts endorsing this instance of a post.
    protected boolean isEmptyPost;
    protected int commentCount;
    protected boolean commentCountUptoDate;

    /**
     * Creates an instance of the Post object. Assigns a message, an ID and an
account to the post object.
     * @param account Account that is creating the post
     * @param message Text contained in the post
     */
    public Post(Account account, String message){
        this.message=message;
        this.account=account;
        postID=nextID++;
        isEmptyPost = false;
        commentCountUptoDate = false;
    }

    /**
     * Returns the account that created the post
     * @return Account that created the post
     */
    public Account getAccount(){
        return account;
    }
    /**
     * Returns the ID of the post
     * @return Unique ID of post
     */
```

```java
    public int getID(){
        return postID;
    }
    /**
     * Returns the text message that the post contains
     * @return Text that the post contains
     */
    public String getMessage(){
        return message;
    }
    /**
     * Returns the total number of comments directly about the post
     * @return Number of comments the post has
     */
    public int getNumComments(){
        if (!commentCountUptoDate){
            int total = 0;
            for (Comment c : comments){
                if (!c.isEmptyPost()){
                    total +=1;
                }
            }
            commentCount = total; // Cache the calculated value
            commentCountUptoDate = true;
        }
        return commentCount;
    }
    /**
     * Sets the commentCoutUpToDate varaible to false
     */
    public void setCommentCountUptoDateToFalse(){
        commentCountUptoDate = false;
    }
    /**
     * Returns the total number of endorsement about the post
     * @return Number of endorsements about the post
     */
    public int getNumEndorsements(){
        return endorsements.size();
    }

    /**
     * Returns the List of comments directly under the post
     * @return ArrayList of comment objects about the post
     */
    public ArrayList<Comment> getComments() {
        return comments;
    }
```

```
92
93      /**
94       * Returns the List of endorsement posts about the post
95       * @return ArrayList of endorsementPost objects about the post
96       */
97      public ArrayList<EndorsementPost> getEndorsements() {
98          return endorsements;
99      }
100     /**
101      * Returns the next sequential ID that will be assigned to the next post
102 that is created
103      * @return ID to be used by the next post that is created
104      */
105     public static int getNextId(){
106         return nextID;
107     }
108
109     /**
110      * States whether the post is an empty post. An empty post is a post that
111      * has been deleted so no longer has a message
112      * @return Boolean value stating whether the post is empty or not
113      */
114     public boolean isEmptyPost(){
115         return isEmptyPost;
116     }
117
118     /**
119      * Sets the next sequential ID, that will be assinged to the next post
120 that is created, to the value
121      * passed in
122      * @param id The value to assign to nextID
123      */
124     public static void setNextId(int id){
125         nextID=id;
126     }
127     /**
128      * Updates the message of the post to provided message
129      * @param message New text that the post object shows when printed
130      */
131     public void setMessage(String message){
132         this.message = message;
133     }
134     /**
135      * Sets the post to be an empty post. If a post is deleted, it is set to
136 empty. Its message is updated
137      * to state that it has been deleted. Comments are still linked to the
138 post
139      */
```

```
140      public void setPostToEmpty(){
141          this.isEmptyPost = true;
142          this.account = null;
143          this.message = "The original content was removed from the system and
144  is no longer available.";
145      }
146      /**
147       * Restets the static ID count of the Post class so that the next post
148  created will have an ID of 1
149       */
150      public static void resetIdCount(){
151          nextID = 1;
152      }
153      /**
154       * Add a comment to the list of comments about the post
155       * @param c Comment object about the post
156       */
157      public void addComment(Comment c){
158          comments.add(c);
159          commentCountUptoDate = false;
160      }
161      /**
162       * Add an endorsement post to the list of endorsements about the post
163       * @param e Endorsement object about the post
164       */
165      public void addEndorsementPost(EndorsementPost e){
166          endorsements.add(e);
167      }
168
169      /**
170       * Removes a single EndorsementPost object from the list of edorsements
171  stored in a post object
172       * @param e The EndorsementPost object that is being removed
173       */
174      public void removeEndorsement(EndorsementPost e){
175          endorsements.remove(e);
176      }
177      /**
178       * Clears the list of endorsements stored in the post Object
179       */
180      public void clearEndorsements(){
181          this.endorsements.clear();
182      }
183
184      /**
185       * Checks that the message meets the requirements to be posted. The mes-
186  sage must be
187       * less than 100 characters and must not be empty
```

```
188         * @param message Text that is being checked
189         * @throws InvalidPostException If the message is empty or if it is longer
190     than 100 characters
191         */
192        public static void validateMessage(String message) throws InvalidPostEx-
193     ception{
194            if (message.length()>100 || message.isEmpty()){
195                throw new InvalidPostException();
196            }
197        }
198        /**
199         * Searches through all of the posts stored in accounts and all of the
200     empty posts stored in an emptyPostList.
201         * Returns the Post object that has the given ID
202         * @param id ID of the post to be returned
203         * @param accountList List of accounts stored in SocialMedia
204         * @param emptyPostList List of empty posts
205         * @return Post that has the ID that is passed in
206         * @throws PostIDNotRecognisedException There is no post that has the
207     given ID
208         */
209        public static Post findPostByID(int id, ArrayList<Account> accountList,
210     ArrayList<Post> emptyPostList) throws PostIDNotRecognisedException{
211            for (Account a : accountList){ // Searches through posts stored in ac-
212     counts
213                for(Post p : a.getPosts()){
214                    if (p.getID() == id){
215                        return p;
216                    }
217                }
218            }
219            for (Post p : emptyPostList){ // Searches through posts stored in emp-
220     tyPostList
221                if (p.getID() == id){
222                    return p;
223                }
224            }
225            throw new PostIDNotRecognisedException();
226        }
227        /**
228         * Searches through all of the posts stored in accounts and returns the
229     Post object that has the given ID
230         * @param id ID of the post to be returned
231         * @param accountList List of accounts stored in SocialMedia
232         * @return Post that has the ID that is passed in
233         * @throws PostIDNotRecognisedException There is no post that has the
234     given ID
235         */
```

```
236      public static Post findPostByID(int id, ArrayList<Account> accountList)
237  throws PostIDNotRecognisedException{
238          for (Account a : accountList){ // Searches through posts stored in ac-
239  counts
240              for(Post p : a.getPosts()){
241                  if (p.getID() == id){
242                      return p;
243                  }
244              }
245
246          }
247          throw new PostIDNotRecognisedException();
248      }
249
250      /**
251       * Formats the text to be printed by the showPostChildrenDetails method.
252  Adds |> before the first line
253       * and some number of indents before each new line so that the message is
254  indented when printed. If it is
255       * the first comment of a post a blank line containing a single | is added
256  about. So if the text passed
257       *  in is the first comment of a post it returns:
258       *
259       * <pre>
260       *     |
261       *     | > ID: 1
262       *     Account: user1
263       *     No. endorsements: 0 | No. comments: 0
264       *     This is the first comment
265       * </pre>
266       *
267       * @param message The description of the post that is being formatted
268       * @param indentLevel The number of times the message is indented
269       * @param isFirstComment Boolean value stating whether the comment is the
270  first comment of a post
271       * @return New string which is the indented message with the appropriate
272  symbols | > before the first line
273       */
274      public static StringBuilder formatMessage(String message, int indentLevel,
275  boolean isFirstComment){
276          StringBuilder sb = new StringBuilder();
277          boolean isFirstLine = true;
278          for(String line : message.split("\n")){ //Splits message into individ-
279  ual lines
280              if(isFirstLine){
281                  String firstLine = "";
282                  if (indentLevel != 0 && isFirstComment){
```

```
283                     firstLine += "    ".repeat(indentLevel-1)+ "|\n"; // Add
284 empty line with |
285                 }
286                 sb.append(firstLine+"    ".repeat(indentLevel-1)+"| >
287 "+line+"\n"); //Indent line and add '| >' before first line
288                 isFirstLine = false;
289             }
290             else{ // not first line
291                 sb.append("    ".repeat(indentLevel)+line+"\n"); //Indent line
292             }
293         }
294         return sb;
295     }
296     /**
297      * Returns a string containing the information about a post. This includes
298 the ID of the post, the account
299      * that the post belongs to and the number of endorsements and number of
300 comments that the post has. It will
301      * return a string in the form below.
302      * <pre>
303      * ID: 1
304      * Account: user1
305      * No. endorsements: 2 | No. comments: 1
306      * This is a post
307      * </pre>
308      * <p>
309      * An empty post, which has been deleted will display a message stating
310 that it has been deleted.
311      * <pre>
312      * ----------------------------------------------------------------------
313 -----
314      * The original content was removed from the system and is no longer
315 available.
316      * ----------------------------------------------------------------------
317 -----
318      * </pre>
319      * @return String of information about the post
320      */
321     @Override
322     public String toString(){
323         if (isEmptyPost){
324             return "-".repeat(message.length())+"\n"+message+"\n"+"-".re-
325 peat(message.length())+"\n";
326         }
327         return "ID: "+ postID +
328         "\nAccount: "+ account.getHandle()+
329         "\nNo. endorsements: "+ getNumEndorsements()+" | No. comments: "+
330 getNumComments()+ "\n" +
```

```
331             message;
332         }
333     }
```

Comment.java

```java
package socialmedia;

public class Comment extends Post{
    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

```
1   package socialmedia;
2
3   public class EndorsementPost extends Post{
4
5       private Post referencePost;
6       /**
7        * Creates an instance of a EndorsementPost object, by first calling the
8   post
9        * superclass constructor and then assigning a value for reference post.
10       * @param account the account that is posting the endorsement.
11       * @param message The text shown in the post body. This will contain the
12  original post message
13       * @param referencePost The post that is being endorsed
14       */
15      public EndorsementPost(Account account, String message, Post reference-
16  Post) {
17          super(account, message);
18          this.referencePost = referencePost;
19          referencePost.addEndorsementPost(this);
20          referencePost.getAccount().setEndorsementCountUpToDate-
21  ToFalse();//postingAccount.setEndorsementCountUpToDateToFalse();
22          account.addPost(this);
23      }
24      /**
25       * Returns the post object that the endorsement referes to
26       * @return Post that endorsement referes to
27       */
28      public Post getReferencePost(){
29          return this.referencePost;
30      }
31  }
```