

1 SocialMedia.java

```
1 package socialmedia;
2
3 import java.util.ArrayList;
4 import java.io.FileInputStream;
5 import java.io.FileOutputStream;
6 import java.io.IOException;
7 import java.io.ObjectInputStream;
8 import java.io.ObjectOutputStream;
9
10 /**
11  * An implementor of the SocialMediaPlatform interface
12  *
13  * @author 7720033503 720040807
14  */
15 public class SocialMedia implements SocialMediaPlatform {
16     ArrayList<Account> listOfAccounts = new ArrayList<Account>();
17     ArrayList<Post> listOfPosts = new ArrayList<Post>();
18     Post emptyPost = new Post(-1, "The original content was removed from the system and is no longer
19         available.");
20     // emptyPost has ID 0
21
22     @Override
23     public int createAccount(String handle) throws IllegalHandleException, InvalidHandleException {
24         return createAccount(handle, "");
25     }
26
27     @Override
28     public int createAccount(String handle, String description) throws IllegalHandleException,
29         InvalidHandleException {
30         if (isValidHandle(handle)) {
31             Account temporary = new Account(handle, description);
32             listOfAccounts.add(temporary);
33             return temporary.getID();
34         }
35         return 0;
36     }
37
38     /**
39     * Check if a handle is valid
40     *
41     * @param handle - the handle to check
42     * @return true if the handle is valid
43     * @throws IllegalHandleException - if the handle is already in use
44     * @throws InvalidHandleException - if the handle does not follow valid handle
45     *         rules
46     */
47     private boolean isValidHandle(String handle) throws IllegalHandleException, InvalidHandleException {
48         if ((handle == null) || (handle.length() > 30) || (isWhiteSpace(handle))) {
49             throw new InvalidHandleException();
50         } else if (handleAlreadyExists(handle)) {
51             throw new IllegalHandleException();
52         }
53     }
54 }
```

```

51     } else {
52         return true;
53     }
54
55 }
56
57 /**
58  * Checks if a handle contains a whitespace character
59  *
60  * @param handle - the handle to check
61  * @return boolean - true if there is a whitespace, false if there is not
62  */
63 private boolean isWhiteSpace(String handle) {
64     for (int i = 0; i < handle.length(); i++) {
65         if (handle.charAt(i) == ' ') {
66             return true;
67         }
68     }
69     return false;
70 }
71
72 /**
73  * Check if a handle already exists
74  *
75  * @param handle - the handle to check
76  * @return a boolean - true if the handle exists, false if the handle does not
77  *         exist
78  */
79 private boolean handleAlreadyExists(String handle) {
80     try {
81         findAccountIndex(handle);
82         return true;
83     } catch (HandleNotRecognisedException e) {
84         return false;
85     }
86 }
87
88 /**
89  * finds the index of an account in listOfAccounts from its handle
90  *
91  * @param handle - the handle of the account
92  * @return the index of the account
93  * @throws HandleNotRecognisedException - if the handle does not currently
94  *         belong to any account
95  */
96 private int findAccountIndex(String handle) throws HandleNotRecognisedException {
97     // linear search
98     System.out.println(handle);
99     for (int i = 0; i < listOfAccounts.size(); i++) {
100         System.out.println(listOfAccounts.get(i).getHandle());
101         if (handle.equals(listOfAccounts.get(i).getHandle())) {
102             return i;
103         }
104     }
105     throw new HandleNotRecognisedException();

```

```

106     }
107
108     @Override
109     public void removeAccount(String handle) throws HandleNotRecognisedException {
110         int index = findAccountIndex(handle);
111         int accountID = listOfAccounts.get(index).getID();
112         listOfAccounts.remove(index);
113         int counter = 0;
114         while (counter < listOfPosts.size()) {
115             if (listOfPosts.get(counter).getAuthorID() == accountID) {
116                 try {
117                     deletePost(listOfPosts.get(counter).getID());
118                 } catch (PostIDNotRecognisedException e) {
119                     assert (false); // should never happen because of how we found the IDs
120                 }
121             } else {
122                 counter++;
123             }
124         }
125     }
126
127     @Override
128     public void removeAccount(int ID) throws AccountIDNotRecognisedException {
129         int index = -1;
130         // find index of the account
131         for (int i = 0; i < listOfAccounts.size(); i++) {
132             if (listOfAccounts.get(i).getID() == ID) {
133                 index = i;
134             }
135         }
136         if (index == -1) {
137             throw new AccountIDNotRecognisedException();
138         }
139         listOfAccounts.remove(index);
140         int counter = 0;
141         while (counter < listOfPosts.size()) {
142             if (listOfPosts.get(counter).getAuthorID() == ID) {
143
144                 try {
145                     deletePost(listOfPosts.get(counter).getID());
146                 } catch (PostIDNotRecognisedException e) {
147                     assert (false); // should never happen because of how we found the IDs
148                 }
149             } else {
150                 counter++; // after because indices will change as items are removed
151             }
152         }
153     }
154
155     @Override
156     public void updateAccountDescription(String handle, String description) throws
        HandleNotRecognisedException {
157         int index = findAccountIndex(handle);
158         if (index == -1) {
159             throw new HandleNotRecognisedException();

```

```

160     }
161     listOfAccounts.get(index).setDescription(description);
162     assert (listOfAccounts.get(index).getDescription() == description) : "description not set properly";
163 }
164
165 @Override
166 public void changeAccountHandle(String oldHandle, String newHandle)
167     throws HandleNotRecognisedException, IllegalHandleException, InvalidHandleException {
168     if (isValidHandle(newHandle)) {
169         int index = findAccountIndex(oldHandle);
170         if (index == -1) {
171             throw new HandleNotRecognisedException();
172         }
173         listOfAccounts.get(index).setHandle(newHandle);
174     }
175 }
176
177 @Override
178 public int createPost(String handle, String message) throws HandleNotRecognisedException,
179     InvalidPostException {
180     int index = findAccountIndex(handle);
181     if (index == -1) {
182         throw new HandleNotRecognisedException();
183     }
184     if (isValidMessage(message) == false) {
185         throw new InvalidPostException();
186     }
187     // use a temporary post before adding it to the list
188     Post tempPost = new Post(listOfAccounts.get(index).getID(), message);
189     listOfPosts.add(tempPost);
190     return tempPost.getID();
191 }
192
193 /**
194  * Check if a message is valid within the system
195  *
196  * @param message - the message to check
197  * @return boolean - true if the message is valid, false if the message is not
198  *                 valid
199  */
200 private boolean isValidMessage(String message) {
201     if (message.length() > 100) {
202         return false;
203     } else if (message == "") {
204         return false;
205     }
206     return true;
207 }
208
209 @Override
210 public int getNumberOfAccounts() {
211     return listOfAccounts.size();
212 }
213
214 /**

```

```

214     * finds the index of a post in listOfPosts from its ID
215     *
216     * @param id - the id of the post
217     * @return the index of the post
218     * @throws PostIDNotRecognisedException - if no post has that ID
219     */
220     private int findPostIndex(int id) throws PostIDNotRecognisedException {
221         int counter = 0;
222         while (counter < listOfPosts.size()) {
223             if (listOfPosts.get(counter).getID() == id) {
224                 return counter;
225             }
226             counter++;
227         }
228         throw new PostIDNotRecognisedException();
229     }
230
231     @Override
232     public int endorsePost(String handle, int id)
233         throws HandleNotRecognisedException, PostIDNotRecognisedException, NotActionablePostException {
234         int accountIndex = findAccountIndex(handle); // throws HandleNotRecognisedException
235         int postIndex = findPostIndex(id); // throws PostIDNotRecognisedException
236         if (listOfPosts.get(postIndex) instanceof Endorsement) {
237             throw new NotActionablePostException();
238         }
239         int accountID = listOfAccounts.get(accountIndex).getID();
240         String endorsedAccountHandle =
241             listOfAccounts.get(listOfPosts.get(postIndex).getAuthorID()).getHandle();
242         // build endorsement message
243         String message = "EP@" + endorsedAccountHandle + ": " + listOfPosts.get(postIndex).getMessage();
244         Endorsement tempEndorsement = new Endorsement(accountID, message, id);
245         listOfPosts.add(tempEndorsement);
246         return tempEndorsement.getID();
247     }
248
249     @Override
250     public int commentPost(String handle, int id, String message)
251         throws HandleNotRecognisedException, PostIDNotRecognisedException, NotActionablePostException,
252         InvalidPostException {
253         int accountIndex = findAccountIndex("my_second_handle"); // throws HandleNotRecognisedException
254         int postIndex = findPostIndex(id); // throws PostIDNotRecognisedException
255         if (listOfPosts.get(postIndex) instanceof Endorsement) {
256             throw new NotActionablePostException();
257         }
258         if (!isValidMessage(message)) {
259             throw new InvalidPostException();
260         }
261         Comment tempComment = new Comment(listOfAccounts.get(accountIndex).getID(), message, id);
262         listOfPosts.add(tempComment);
263         return tempComment.getID();
264     }
265
266     @Override
267     public void deletePost(int id) throws PostIDNotRecognisedException {
268         int counter = 0;

```

```

268     while (counter < listOfPosts.size()) {
269         // delete endorsements related to the post
270         if (listOfPosts.get(counter) instanceof Endorsement) {
271             Endorsement tempEndorsement = (Endorsement) listOfPosts.get(counter);
272             if (tempEndorsement.getReferenceID() == id) {
273                 listOfPosts.remove(counter);
274                 counter--;
275             }
276         } // set comments to point to emptyPost
277         else if (listOfPosts.get(counter) instanceof Comment) {
278             Comment tempComment = (Comment) listOfPosts.get(counter);
279             if (tempComment.getReferenceID() == id) {
280                 tempComment.zeroReferenceID();
281             }
282         }
283         counter++;
284     }
285
286     // delete post
287     int index = findPostIndex(id);
288     listOfPosts.remove(index);
289
290 }
291
292 /**
293  * get the number of endorsements that a particular post has
294  *
295  * @param id - the id of the post
296  * @return the number of endorsements that it has
297  */
298 private int getNumEndorsements(int id) {
299     int counter = 0; // number of endorsements
300     // iterate through the posts
301     for (int i = 0; i < listOfPosts.size(); i++) {
302         // check if it's an endorsement
303         if (listOfPosts.get(i) instanceof Endorsement) {
304             Endorsement tempEndorsement = (Endorsement) listOfPosts.get(i);
305             // check if it points to the post
306             if (tempEndorsement.getReferenceID() == id) {
307                 counter++;
308             }
309         }
310     }
311     return counter;
312 }
313
314 /**
315  * get the number of comments that a particular post has
316  *
317  * @param id - the id of the post
318  * @return the number of comments
319  */
320 private int getNumComments(int id) {
321     int counter = 0; // the number of comments
322     // iterate through the list of posts

```

```

323     for (int i = 0; i < listOfPosts.size(); i++) {
324         // check if it's a comment
325         if (listOfPosts.get(i) instanceof Comment) {
326             Comment tempComment = (Comment) listOfPosts.get(i);
327             // check if it points to the post
328             if (tempComment.getReferenceID() == id) {
329                 counter++;
330             }
331         }
332     }
333     return counter;
334 }
335
336 @Override
337 public String showAccount(String handle) throws HandleNotRecognisedException {
338     Account thisAccount = listOfAccounts.get(findAccountIndex(handle));
339     // format string
340     String outputString = "ID: " + Integer.toString(thisAccount.getID()) + "\n"
341         + "Handle: " + handle + "\n"
342         + "Description: " + thisAccount.getDescription() + "\n";
343     // num posts + endorsements on posts
344     int postCount = 0;
345     int endorseCount = 0;
346     for (int i = 0; i < listOfPosts.size(); i++) {
347         if (listOfPosts.get(i).getAuthorID() == thisAccount.getID()) {
348             postCount++;
349             endorseCount += getNumEndorsements(listOfPosts.get(i).getID());
350         }
351     }
352     // back to formatting string
353     outputString += "Post count: " + Integer.toString(postCount) + "\n"
354         + "Endorse count: " + Integer.toString(endorseCount) + "\n";
355     return outputString;
356 }
357
358 @Override
359 public int getTotalEndorsmentPosts() {
360     int counter = 0;
361     for (int i = 0; i < listOfPosts.size(); i++) {
362         if (listOfPosts.get(i) instanceof Endorsement) {
363             counter++;
364         }
365     }
366     return counter;
367 }
368
369 @Override
370 public int getTotalCommentPosts() {
371     int counter = 0;
372     for (int i = 0; i < listOfPosts.size(); i++) {
373         if (listOfPosts.get(i) instanceof Comment) {
374             counter++;
375         }
376     }
377     return counter;

```

```

378     }
379
380     @Override
381     public int getTotalOriginalPosts() {
382         int counter = 0;
383         for (int i = 0; i < listOfPosts.size(); i++) {
384             // check if it is (not endorsement) and (not comment)
385             if (!(listOfPosts.get(i) instanceof Endorsement) && !(listOfPosts.get(i) instanceof Comment)) {
386                 counter++;
387             }
388         }
389         return counter;
390     }
391
392     @Override
393     public void erasePlatform() {
394         listOfAccounts.clear();
395         listOfPosts.clear();
396     }
397
398     @Override
399     public String showIndividualPost(int id) throws PostIDNotRecognisedException {
400         // find the post
401         Post thisPost;
402         if (id == 0) { // the empty post
403             thisPost = emptyPost;
404         } else {
405             thisPost = listOfPosts.get(findPostIndex(id));
406         }
407         return showPostFormat(id, thisPost, "");
408     }
409
410     /**
411      * Format the string for showIndividualPost or showPostChildrenDetails
412      *
413      * @param id - the id of the post
414      * @param thisPost - the post itself (Post object)
415      * @param spaces - this is here to be used when called from
416      *                 showPostChildrenDetails, it is set to "" when called from
417      *                 showIndividualPost
418      * @return the formatted string
419      */
420     private String showPostFormat(int id, Post thisPost, String spaces) {
421         // format string
422         String outputString = "ID: " + Integer.toString(id) + "\n"
423             + spaces + "Account: " + listOfAccounts.get(thisPost.getAuthorID()).getHandle() + "\n"
424             + spaces + "No. endorsements: " + Integer.toString(getNumEndorsements(id))
425             + spaces + " | No. comments: " + Integer.toString(getNumComments(id)) + "\n"
426             + spaces + thisPost.getMessage() + "\n";
427         return outputString;
428     }
429
430     @Override
431     public StringBuilder showPostChildrenDetails(int id)
432         throws PostIDNotRecognisedException, NotActionablePostException { // recursive

```



```

433     int index = findPostIndex(id);
434     if (listOfPosts.get(index) instanceof Endorsement) { // Check the post is not an endorsement
435         throw new NotActionablePostException();
436     }
437     StringBuilder str = new StringBuilder();
438     String spaces = "";
439     showChildrenRecursive(str, id, spaces);
440     return str;
441 }
442
443 /**
444  * The recursive algorithm that forms the basis of showPostChildrenDetails
445  * It has to be recursive so that spaces can be incremented depending on the
446  * level of comment
447  *
448  * @param str - the StringBuilder object that is being added onto
449  * @param id - the id of the post that is currently being looked at
450  * @param spaces - the string of spaces that go before the post
451  * @throws PostIDNotRecognisedException - if the ID isn't an existing post. this
452  * should never occur.
453  */
454 private void showChildrenRecursive(StringBuilder str, int id, String spaces)
455     throws PostIDNotRecognisedException {
456     str.append(showPostFormat(id, listOfPosts.get(findPostIndex(id)), spaces));
457     spaces += "  ";
458     // iterate through listOfPosts to find all comments pointing to the post
459     for (int i = 0; i < listOfPosts.size(); i++) {
460         if (listOfPosts.get(i) instanceof Comment) {
461             Comment tempComment = (Comment) listOfPosts.get(i);
462             if (tempComment.getReferenceID() == id) {
463                 // tempComment points to the post
464                 str.append(spaces + "\\n");
465                 str.append(spaces + "| > ");
466                 // call this function with tempComment
467                 showChildrenRecursive(str, tempComment.getID(), spaces);
468             }
469         }
470     }
471 }
472
473 @Override
474 public int getMostEndorsedPost() {
475     int maxEndorsement = 0;
476     int maxEndorsedID = 0;
477     int endorsementCount;
478     int thisPostID;
479     // iterate through listOfPosts in order to find the maximum number of
480     // endorsements
481     for (int i = 0; i < listOfPosts.size(); i++) {
482         thisPostID = listOfPosts.get(i).getID();
483         endorsementCount = getNumEndorsements(thisPostID);
484         if (maxEndorsement < endorsementCount) {
485             maxEndorsedID = thisPostID;
486             maxEndorsement = endorsementCount;
487         }
488     }
489 }

```

```

488     }
489     return maxEndorsedID;
490 }
491
492 @Override
493 public int getMostEndorsedAccount() {
494     int maxEndorsement = 0;
495     int maxEndorsedID = 0;
496     int endorsementCount;
497     int thisAccountID;
498     // iterate through listOfAccounts in order to find the maximum number of
499     // endorsements
500     for (int i = 0; i < listOfAccounts.size(); i++) {
501         endorsementCount = 0;
502         thisAccountID = listOfAccounts.get(i).getID();
503         for (int j = 0; j < listOfPosts.size(); j++) {
504             if (listOfPosts.get(j).getAuthorID() == thisAccountID) {
505                 endorsementCount++;
506                 endorsementCount += getNumEndorsements(listOfPosts.get(j).getID());
507             }
508             if (endorsementCount > maxEndorsement) {
509                 maxEndorsedID = thisAccountID;
510                 maxEndorsement = endorsementCount;
511             }
512         }
513     }
514     return maxEndorsedID;
515 }
516
517 @Override
518 public void savePlatform(String filename) throws IOException {
519     ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(filename));
520     out.writeObject(listOfPosts);
521     out.writeObject(listOfAccounts);
522     out.close();
523 }
524
525 @Override
526 public void loadPlatform(String filename) throws IOException, ClassNotFoundException {
527     erasePlatform();
528     ObjectInputStream in = new ObjectInputStream(new FileInputStream(filename));
529     listOfPosts = (ArrayList) in.readObject();
530     listOfAccounts = (ArrayList) in.readObject();
531     // set the counter for the IDs for Post and Account from the ID of the latest in
532     // the list
533     Post.setIDOnLoad(listOfPosts.get(listOfPosts.size() - 1).getID());
534     Account.setIDOnLoad(listOfAccounts.get(listOfAccounts.size() - 1).getID());
535     in.close();
536 }
537 }

```

2 Account.java

```

1 package socialmedia;

```

```

2
3 import java.io.Serializable;
4
5 /**
6  * Account class
7  * Attributes:
8  * static counter - used to count the IDs to ensure they are unique
9  * ID - unique int for each Account (starts from 0)
10 * handle - unique account handle
11 * desc - account description
12 *
13 * @author - 7720033503 720040807
14 */
15
16 public class Account implements Serializable {
17     protected static int counter = 0;
18     private int ID;// unique
19     // ID starts from 0 for Account class
20     private String handle;// unique also
21     private String description;
22
23     /**
24      * Account creation method
25      *
26      * @param handle - the unique handle of the account
27      * @param desc - the description of the account, may be an empty string
28      */
29     public Account(String handle, String desc) {
30         this.ID = counter;
31         counter++;
32         this.handle = handle;
33         this.description = desc;
34     }
35
36     // setters and getters *****
37
38     /**
39      * set counter (which the IDs count from), used only when loading
40      *
41      * @param maxID - ID of the last created account. next ID will be maxID+1
42      */
43     public static void setIDOnLoad(int maxID) {
44         counter = maxID + 1;
45     }
46
47     /**
48      * getter for ID
49      *
50      * @return ID
51      */
52     public int getID() {
53         return ID;
54     }
55
56     /**

```

```

57     * getter for handle
58     *
59     * @return handle
60     */
61     public String getHandle() {
62         return handle;
63     }
64
65     /**
66     * setter for handle
67     *
68     * @param newHandle - must be a valid handle
69     */
70     public void setHandle(String newHandle) { // not validated here
71         handle = newHandle;
72     }
73
74     /**
75     * getter for description
76     *
77     * @return description
78     */
79     public String getDescription() {
80         return description;
81     }
82
83     /**
84     * setter for description
85     *
86     * @param newDescription
87     */
88     public void setDescription(String newDescription) {
89         description = newDescription;
90     }
91 }

```

3 Post.java

```

1  package socialmedia;
2
3  import java.io.Serializable;
4
5  /**
6   * Post class
7   * Attributes:
8   * static counter - used to count the IDs to ensure they are unique
9   * ID - unique int for each Post (starts from 0)
10  * message - must be <=100chars
11  * authorID - ID of the account that the post is associated with
12  *
13  * @author - 7720033503 720040807
14  */
15
16  public class Post implements Serializable {

```

```

17     protected static int counter = 0;
18     protected int ID;// unique
19     // IDs for normal posts start at 1 - 0 is emptyPost
20     protected String message;// <=100chars
21     protected int authorID;
22
23     public Post(int authorID, String message) { // valid message - validated in createPost
24         this.ID = counter;
25         counter++;
26         this.authorID = authorID;
27         this.message = message;
28     }
29
30     // setters and getters *****
31
32     /**
33      * set counter (which the IDs count from), used only when loading
34      *
35      * @param maxID - ID of the last created account. next ID will be maxID+1
36      */
37     public static void setIDOnLoad(int maxID) {
38         counter = maxID + 1;
39     }
40
41     /**
42      * getter for ID
43      *
44      * @return ID
45      */
46     public int getID() {
47         return ID;
48     }
49
50     /**
51      * getter for authorID
52      *
53      * @return authorID
54      */
55     public int getAuthorID() {
56         return authorID;
57     }
58
59     /**
60      * getter for message
61      *
62      * @return message
63      */
64     public String getMessage() {
65         return message;
66     }
67
68 }

```

4 Comment.java

```
1 package socialmedia;
2
3 /**
4  * Comment inherits from Post
5  * Has additional attribute:
6  * referenceID - the ID of the post that the comment is referencing
7  */
8 public class Comment extends Post {
9     protected int referenceID;// post that the comment is referencing
10
11     /**
12      * Constructor for Comment
13      *
14      * @param accountID - the account making the comment
15      * @param message - the message that the comment will carry
16      * @param referenceID - the ID of the post that the comment is referencing
17      */
18     public Comment(int accountID, String message, int referenceID) {
19         super(accountID, message);
20         this.referenceID = referenceID;
21     }
22
23     /**
24      * getter for referenceID
25      *
26      * @return referenceID
27      */
28     public int getReferenceID() {
29         return referenceID;
30     }
31
32     /**
33      * set the referenceID to 0
34      * this points it to an empty post
35      * to be called in deletePost
36      */
37     public void zeroReferenceID() {
38         referenceID = 0;
39     }
40 }
```

5 Endorsement.java

```
1 package socialmedia;
2
3 /**
4  * Endorsement inherits from Post
5  * Has additional attribute:
6  * referenceID - the ID of the post that the endorsement is referencing
7  */
8 public class Endorsement extends Post {
9     protected int referenceID;// post that the endorsement is referencing
```

```

10
11  /**
12   * Constructor for Endorsement
13   *
14   * @param accountID - the account making the endorsement
15   * @param message   - the message that the endorsement will carry
16   * @param referenceID - the ID of the post that the endorsement is referencing
17   */
18  public Endorsement(int accountID, String message, int referenceID) {
19      super(accountID, message);
20      this.referenceID = referenceID;
21  }
22
23  /**
24   * getter for referenceID
25   *
26   * @return referenceID
27   */
28  public int getReferenceID() {
29      return referenceID;
30  }
31  }

```