```java
package socialmedia;

import socialmedia.socialmedia.BasePost;
import socialmedia.socialmedia.User;
import socialmedia.socialmedia.Post;
import socialmedia.socialmedia.Comment;
import socialmedia.socialmedia.Endorsement;
import socialmedia.socialmedia.interfaces.Interactable;
import socialmedia.socialmedia.excepts.*;
import socialmedia.socialmedia.interfaces.SocialMediaPlatform;

import java.io.*;
import java.util.*;


public class SocialMedia implements SocialMediaPlatform {

    // Stores accounts with in the form (ID, handle).
    HashMap<Integer, User> accounts = new HashMap<>();

    // Stores all posts that inherit from BasePost (i.e. BasePost, Post, Comment, Endorsement).
    ArrayList<BasePost> posts = new ArrayList<>();

    @Override
    public int createAccount(String handle) throws IllegalHandleException, InvalidHandleException {
        // Error handling
        if (accounts.containsKey(Objects.hash(handle)))
            throw new IllegalHandleException();
        if (handle.length() == 0 || handle.length() > 30 || handle.contains(" "))
            throw new InvalidHandleException();

        // Initialise
        User usr = new User(handle);
        accounts.put(Objects.hash(handle), usr);
        return Objects.hash(handle);
    }

    @Override
    public int createAccount(String handle, String description) throws IllegalHandleException, InvalidHandleException {
        // Error handling
        if (accounts.containsKey(Objects.hash(handle)))
            throw new IllegalHandleException();
        if (handle.length() == 0 || handle.length() > 30 || handle.contains(" "))
            throw new InvalidHandleException();

        // Initialise
        User usr = new User(handle, description);
        accounts.put(Objects.hash(handle), usr);
        return Objects.hash(handle);
    }

    @Override
    public void removeAccount(int id) throws AccountIDNotRecognisedException {
        if (accounts.remove(id) == null) throw new
```

```
 54 AccountIDNotRecognisedException();
 55
 56         // Remove all related posts after account has been removed
 57         posts.stream().filter(x -> x.getAuthorID() == id).forEach(x -> {
 58             deleteAllRelatedPosts(x, posts);
 59             posts.remove(x);
 60         });
 61     }
 62
 63     @Override
 64     public void removeAccount(String handle) throws
    HandleNotRecognisedException {
 65         if (accounts.remove(Objects.hash(handle)) == null) throw new
    HandleNotRecognisedException();
 66
 67         // Remove all related posts after account has been removed
 68         posts.stream().filter(x -> x.getAuthorID() == Objects.hash(handle)).
    forEach(x -> {
 69             deleteAllRelatedPosts(x, posts);
 70             posts.remove(x);
 71         });
 72     }
 73
 74     @Override
 75     public void changeAccountHandle(String oldHandle, String newHandle)
 76             throws HandleNotRecognisedException, IllegalHandleException,
    InvalidHandleException {
 77
 78         User user = accounts.get(Objects.hash(oldHandle));
 79
 80         // Error handling
 81         if (Objects.equals(user, null))
 82             throw new HandleNotRecognisedException();
 83         if (accounts.containsKey(Objects.hash(newHandle)))
 84             throw new IllegalHandleException();
 85         if (newHandle.length() == 0 || newHandle.length() > 30 || newHandle.
    contains(" "))
 86             throw new InvalidHandleException();
 87
 88         // Replacing user in the database to accommodate for the new handle (
    since our hash has been changed).
 89         accounts.remove(Objects.hash(oldHandle), user);
 90         user.setHandle(newHandle);
 91         accounts.put(Objects.hash(newHandle), user);
 92
 93         // Replace all authorIDs with the new handle's ID.
 94         posts.stream()
 95                 .filter(x -> x.getAuthorID() == Objects.hash(oldHandle))
 96                 .forEach(x -> x.setAuthorID(Objects.hash(newHandle)));
 97     }
 98
 99     @Override
100     public void updateAccountDescription(String handle, String description)
    throws HandleNotRecognisedException {
101         User user = accounts.get(Objects.hash(handle));
102
103         // Error handling
104         if (Objects.equals(user, null)) throw new HandleNotRecognisedException
    ();
```

```
105
106         // Initialise
107         user.setDescription(description);
108     }
109
110     @Override
111     public String showAccount(String handle) throws
    HandleNotRecognisedException {   // Not complete
112         User user = accounts.get(Objects.hash(handle));
113
114         // Error handling
115         if (Objects.equals(user, null))
116             throw new HandleNotRecognisedException();
117
118         // Get all posts where the authorID matches that of the given user's
119         List<BasePost> userPosts = posts.stream().filter(x -> x.getAuthorID
    () == user.getId()).toList();
120         int endorsementCount = 0;
121
122         // For each post, increment endorsementCount by the endorseCount of
    that post.
123         for (BasePost post : userPosts)
124             if (post instanceof Interactable) endorsementCount += ((
    Interactable) post).getEndorseCount();
125
126         // Return a string constructed from the data.
127         return "ID: " + user.getId() + "\n" +
128                 "Handle: " + user.getHandle() + "\n" +
129                 "Description: " + user.getDescription() + "\n" +
130                 "Post Count: " + userPosts.size() + "\n" +
131                 "Endorse Count: " + endorsementCount;
132     }
133
134     @Override
135     public int createPost(String handle, String message) throws
    HandleNotRecognisedException, InvalidPostException {
136         // Error handling
137         if (!accounts.containsKey(Objects.hash(handle)))
138             throw new HandleNotRecognisedException();
139         if (message.length() > 100 || message.length() == 0)
140             throw new InvalidPostException();
141
142         // Initialise Post
143         Post post = new Post(message, Objects.hash(handle));
144         posts.add(post);
145         return post.getId();
146     }
147
148     /**
149      * The method searches through our database and finds the first post with
    the given postID.
150      * @param id ID to search for
151      * @return The post related to that ID, or null if a post was not found.
152      */
153     BasePost findPostById(int id) {
154         // Get all posts with the ID given through the parameter, and if it
    can't find that post, return null.
155         return posts.stream()
156                 .filter(x -> Objects.equals(x.getId(), id))
```

```
157                 .findFirst()
158                 .orElse(null);
159     }
160 
161     @Override
162     public int endorsePost(String handle, int id)
163             throws HandleNotRecognisedException, PostIDNotRecognisedException
    , NotActionablePostException {
164 
165         BasePost post = findPostById(id);
166 
167         // Error handling
168         if (!accounts.containsKey(Objects.hash(handle)))
169             throw new HandleNotRecognisedException();
170         if (Objects.equals(post, null)) {
171             throw new PostIDNotRecognisedException();
172         }
173         if (!(post instanceof Interactable)) {
174             throw new NotActionablePostException();
175         }
176 
177         // Initialise
178         Endorsement endorsement = new Endorsement(post.getMessage(), Objects.
    hash(handle), post.getId());
179         posts.add(endorsement);
180 
181         // Not necessary to increment all endorse counts, so simply increment
    the original post by one.
182         ((Interactable) post).incrementEndorseCount();
183 
184         return endorsement.getId();
185     }
186 
187     @Override
188     public int commentPost(String handle, int id, String message) throws
    HandleNotRecognisedException,
189             PostIDNotRecognisedException, NotActionablePostException,
    InvalidPostException {
190 
191         BasePost post = findPostById(id);
192 
193         // Error handling
194         if (!accounts.containsKey(Objects.hash(handle)))
195             throw new HandleNotRecognisedException();
196         if (Objects.equals(post, null)) {
197             throw new PostIDNotRecognisedException();
198         }
199         if (post instanceof Endorsement && !(post instanceof Comment)) {
200             throw new NotActionablePostException();
201         }
202         if (message.length() > 100 || message.length() == 0)
203             throw new InvalidPostException();
204 
205         // Initialise Comment
206         Comment comment = new Comment(message, Objects.hash(handle), id);
207         posts.add(comment);
208 
209         Interactable pointer = (Interactable) findPostById(comment.
    getOriginalPostID());
```

```
210
211         // Cascade up and increment all parents' comment counts
212         while (pointer != null) {
213             pointer.incrementCommentCount();
214             pointer = (pointer instanceof Comment) ? (Interactable)
    findPostById(((Comment) pointer).getOriginalPostID()) : null;
215         }
216
217         return comment.getId();
218     }
219
220     @Override
221     public void deletePost(int id) throws PostIDNotRecognisedException {
222         BasePost post = findPostById(id);
223
224         if (Objects.equals(post, null)) throw new PostIDNotRecognisedException
    ();
225
226         // Delete all posts that are children of the given post
227         int removedCommentCount = deleteAllRelatedPosts(post, posts);
228
229         // Cascade up and decrement comment counts by the amount of posts that
     were removed by the function
230         // All parents should be either Comments or Posts, so we cast to
    Interactable
231         Interactable pointer;
232         if (post instanceof Comment && removedCommentCount != 0) { // Initial
    check just so if our deleted post was a Post we don't waste time
233             pointer = (Interactable) findPostById(((Comment) post).
    getOriginalPostID());
234
235             while (pointer != null) {
236                 System.out.println("Removing: " + removedCommentCount + " from
     " + ((BasePost) pointer).getId() + "\n");
237
238                 pointer.setCounts(pointer.getCommentCount() -
    removedCommentCount, pointer.getEndorseCount());
239                 pointer = (pointer instanceof Comment) ? (Interactable)
    findPostById(((Comment) pointer).getOriginalPostID()) : null;
240             }
241         }
242     }
243
244     /**
245      * The method uses a depth-first iterative method to erase an element and
    all elements related to it by using a stack.
246      * Not ideal, Big O of O(n + (n^2)^m), however it works for now
247      * @param original Post for which all of its children should be found.
248      * @param list List to be iterated over.
249      * @return The number of comments removed, for use in correcting
    commentCount values.
250      */
251     int deleteAllRelatedPosts(BasePost original, List<BasePost> list) { // Add
     void Predicate later
252         if (!(original instanceof Interactable)) { // If original is an
    Endorsement
253             list.remove(original);
254             return 0;
255         }
```

```
256
257         // Create stack to store "tree" levels
258         Stack<List<Endorsement>> levels = new Stack<>();
259
260         // Prepare container to store number of deleted comments
261         int deletedComments = (original instanceof Comment) ? 1 : 0;
262
263         // Get all children of the original Post
264         List<Endorsement> start = list.stream()
265                 .filter(x -> x instanceof Endorsement)
266                 .map(x -> (Endorsement) x)
267                 .filter(x -> ((Endorsement) x).getOriginalPostID() == original
    .getId())
268                 .toList();
269
270         // Push to stack, and remove it from posts.
271         levels.push(start);
272         list.remove(original);
273
274         // Run until the stack is empty (or, until we can no longer find
    children of any nodes in level).
275         while (!levels.isEmpty()) {
276             var level = levels.pop();
277
278             // For every child in level...
279             for (Endorsement post : level) {
280                 // Get all children of the post
281                 var postsRelated = list.stream()
282                         .filter(x -> x instanceof Endorsement)
283                         .map(x -> (Endorsement) x)
284                         .filter(x -> x.getId() == post.getOriginalPostID())
285                         .toList();
286
287                 // Push all found children to the stack
288                 levels.push(postsRelated);
289
290                 // Remove all posts discovered from the stack and continue to
    the next level
291                 if (post instanceof Comment) deletedComments++;
292                 list.remove(post);
293             }
294         }
295
296         return deletedComments;
297     }
298
299     @Override
300     public String showIndividualPost(int id) throws
    PostIDNotRecognisedException {
301         // Error handling
302         BasePost post = findPostById(id);
303
304         if (Objects.equals(post, null)) throw new PostIDNotRecognisedException
    ();
305
306         // Return formatted string (endorseCount and commentCount are omitted
    if post is an endorsement)
307         return "ID: " + post.getId() + "\n" +
308                 "Account: " + accounts.get(post.getAuthorID()).getHandle() + "
```

```
308 \n" + ((post instanceof Interactable) ?
309                 ("No. endorsements: " + ((Interactable) post).getEndorseCount
    () + " | No. comments: " + ((Interactable) post).getCommentCount())
310                         + "\n" + post.getMessage() : "") ;
311     }
312 
313     @Override
314     public StringBuilder showPostChildrenDetails(int id)
315             throws PostIDNotRecognisedException, NotActionablePostException {
316 
317         BasePost post = findPostById(id);
318         StringBuilder builder = new StringBuilder();
319 
320         // Error handling
321         if (post == null) throw new PostIDNotRecognisedException();
322         if (!(post instanceof Interactable)) throw new
    NotActionablePostException();
323 
324         // Call to recursive function to find all children and build formatted
     string
325         builder.append(showIndividualPost(id)).append("\n");
326         buildChildrenString(post, builder, 0);
327 
328         return builder;
329     }
330 
331     /**
332      * The method builds a string to fit the required format of {@link #
    showPostChildrenDetails(int) showPostChildrenDetails(id)}
333      * by recursively iterating over the children of the target post in a <
    strong>breadth-first manner.</strong>
334      * @param target Post in posts list to be targeted by the recursive
    function.
335      * @param builder StringBuilder reference object to append strings to.
336      * @param indent Starting indent for the printed list.
337      */
338     void buildChildrenString(BasePost target, StringBuilder builder, int
    indent) throws PostIDNotRecognisedException {
339         try {
340             // For every post in our list...
341             for (BasePost post : posts) {
342                 // If the post is a comment and its originalID matches our
    target's ID...
343                 if (post instanceof Comment && ((Comment) post).
    getOriginalPostID() == target.getId()) {
344                     String[] lines = showIndividualPost(post.getId()).split("\
    n");
345 
346                     // Append a string in the required format to the builder
347                     String appString = (((indent == 0) ? "" : "|\n") + "| > "
     + lines[0] + "\n").indent(indent) +
348                             ((lines[1] + "\n") +
349                             (lines[2] + "\n") +
350                             (lines[3])).indent(4 + indent);
351 
352                     builder.append(appString);
353 
354                     // Recursively call this function, incrementing the indent
     by 4 (the tab space taken by "| > ")
```

```
355                     buildChildrenString(post, builder, indent + 4);
356                 }
357             }
358         } catch (PostIDNotRecognisedException e) {
359             throw new PostIDNotRecognisedException();
360         }
361     }
362 
363     @Override
364     public int getNumberOfAccounts() {
365         // TODO Auto-generated method stub
366         return accounts.size();
367     }
368 
369     @Override
370     public int getTotalOriginalPosts() {
371         // TODO Auto-generated method stub
372         return (int) posts.stream()
373                 .filter(x -> x instanceof Post)
374                 .count();
375     }
376 
377     @Override
378     public int getTotalEndorsmentPosts() {
379         // TODO Auto-generated method stub
380         return (int) posts.stream()
381                 .filter(x -> !(x instanceof Interactable))
382                 .count();
383     }
384 
385     @Override
386     public int getTotalCommentPosts() {
387         // TODO Auto-generated method stub
388         return (int) posts.stream()
389                 .filter(x -> x instanceof Comment)
390                 .count();
391     }
392 
393     @Override
394     public int getMostEndorsedPost() {
395         if (posts.size() == 0) return 0;
396 
397         Interactable currentMostEndorsed = new Post("", 0);
398 
399         // For each post in our list, if the post's endorseCount is greater
    then set currentMostEndorsed to post.
400         for (BasePost post : posts) {
401             if (post instanceof Interactable && ((Interactable) post).
    getEndorseCount() > currentMostEndorsed.getEndorseCount()) {
402                 currentMostEndorsed = (Interactable) post;
403             }
404         }
405 
406         return ((BasePost) currentMostEndorsed).getId();
407     }
408 
409     @Override
410     public int getMostEndorsedAccount() {
411         if (posts.size() == 0) return 0;
```

```
412
413         int mostEndorsedAccount = 0;
414         int mostAccountEndorsements = 0;
415
416         // For each user in the database
417         for (Map.Entry<Integer, User> account : accounts.entrySet()) {
418             int accountEndorsements = 0;
419             User user = account.getValue();
420
421             // Loop through all the posts and find all posts related to this
    user, then increment their accountEndorsements
422             // for each post found.
423             for (BasePost post : posts) {
424                 if (post instanceof Interactable && post.getAuthorID() == user
    .getId()) {
425                     accountEndorsements += ((Interactable) post).
    getEndorseCount();
426                 }
427
428             }
429             // If we have found a more endorsed account, replace the ID in
    mostAccountEndorsements with the new one.
430             if (accountEndorsements > mostAccountEndorsements) {
431                 mostAccountEndorsements = accountEndorsements;
432                 mostEndorsedAccount = user.getId();
433             }
434         }
435
436         return mostEndorsedAccount;
437     }
438
439     @Override
440     public void erasePlatform() {
441         // Reset the sequential nextID counter for posts
442         BasePost.resetCounter();
443
444         // Clear both post and account lists
445         posts.clear();
446         accounts.clear();
447     }
448
449     @Override
450     public void savePlatform(String filename) throws IOException {
451         File file = new File(filename);
452
453         // Create writer
454         try (BufferedWriter writer = new BufferedWriter(new FileWriter(file
    ))) {
455             // Create comma seperated line for each post
456             for (BasePost post : posts) {
457                 writer.write(post.toString());
458                 writer.newLine();
459             }
460             // Create comma seperated line for each account
461             for (Map.Entry<Integer, User> account : accounts.entrySet()) {
462                 writer.write(account.getValue().toString());
463                 writer.newLine();
464             }
465             // Finish document with nextId
```

```
466             writer.write("nextId," + BasePost.getCounter());
467         } catch (IOException e) {
468             throw new IOException();
469         }
470     }
471 
472     @Override
473     public void loadPlatform(String filename) throws IOException,
    ClassNotFoundException {
474         // TODO Auto-generated method stub
475         // Get file
476         File file = new File(filename);
477 
478         // Instantiate reader
479         try (BufferedReader reader = new BufferedReader(new FileReader(file
    ))) {
480             // For each line in the file
481             for (String line : reader.lines().toList()) {
482                 // Split file by commas
483                 String[] data = line.split(",");
484                 switch (data[0]) {
485                     // Check first element in comma-separated line to get
    class name
486                     case "Post" -> {
487                         Post post = new Post(data[2], Integer.parseInt(data[3
    ]), Integer.parseInt(data[1]));
488                         post.setCounts(Integer.parseInt(data[4]), Integer.
    parseInt(data[5]));
489                         posts.add(post);
490                     }
491 
492                     case "Comment" -> {
493                         Comment com = new Comment(data[2], Integer.parseInt(
    data[3]), Integer.parseInt(data[1]));
494                         com.setCounts(Integer.parseInt(data[4]), Integer.
    parseInt(data[5]));
495                         posts.add(com);
496                     }
497 
498                     case "Endorsement" -> {
499                         Endorsement end = new Endorsement(data[2], Integer.
    parseInt(data[3]), Integer.parseInt(data[4]),  Integer.parseInt(data[1]));
500                         posts.add(end);
501                     }
502 
503                     case "User" -> {
504                         User usr = null;
505 
506                         // Handling in case no description is found
507                         if (data.length >2){
508                             usr = new User(data[1], data[2]);}
509                         else{
510                             usr = new User(data[1]);
511                         }
512                         accounts.put(Objects.hash(data[1]), usr);
513                     }
514 
515                     case "nextId" -> BasePost.setCounter(Integer.parseInt(data
    [1]));
```

```
516                     case "" -> System.out.println("Empty line");
517
518                     case "Default" -> throw new ClassNotFoundException();
519                 }
520             }
521         } catch (IOException e) {
522             throw new IOException();
523         }
524     }
525
526
527 }
```

```java
package socialmedia.socialmedia;

import java.util.Objects;

public class User {
    private int id;
    private String handle;
    private String description = "";

    public User(String handle){
        this.handle = handle;
        this.id = Objects.hash(handle);
    }
    public User(String handle, String description){
        this.handle = handle;
        this.description = description;
        this.id = Objects.hash(handle);
    }

    /**
     * The method returns the ID of the User object.
     * @return ID of the object.
     */
    public int getId() { return id; }

    /**
     * The method updates the ID of the object using its handle. This typically does nothing, but when the ID is updated,
     * we need to change the ID to match the hash value, and this method handles that.
     */
    void updateId() { id = Objects.hash(handle); }

    /**
     * The method returns the description of the User object.
     * @return Description of the object.
     */
    public String getDescription() {
        return description;
    }

    /**
     * The method returns the handle of the User object.
     * @return Handle of the object.
     */
    public String getHandle() {
        return handle;
    }

    /**
     * The method returns the contents of the class as a string value. Used in this format for saving and loading the
     * platform.
     * @param description Value to set to the description.
     */
    public void setDescription(String description) {
        this.description = description;
    }

```

```
57      /**
58       * The method sets the handle to a new value, then updates the ID of the
    class to match the Objects.hash() result.
59       * @param handle The value to set to the handle.
60       */
61      public void setHandle(String handle) {
62          this.handle = handle;
63          updateId();
64      }
65
66      /**
67       * The method returns the contents of the class as a string value. Used in
     this format for saving and loading the
68       * platform.
69       * @return String containing object's contents in the format "User,handle,
    description".
70       */
71      public String toString(){
72          return "User," + handle + ',' + description;
73      }
74  }
```

```java
package socialmedia.socialmedia;

import socialmedia.socialmedia.interfaces.Interactable;

/**
 * Post is a class that inherits BasePost and implements the interface of Interactable, allowing it to store comment
 * and endorsmeent information.
 * <p></p>
 * Post is inherited by itself to store the functionalities for an original post, as well as the methods of Interactable,
 * without allowing Endorsement to inherit the same functionality, therefore saving space and complexity.
 *
 * @author Daniel Casley, Benjamin Richmond
 * @version 1.0
 */
public class Post extends BasePost implements Interactable {
    public Post(String message, int authorID) {
        id = nextID++;
        this.message = message;
        this.authorID = authorID;
    }
    public Post(String message, int authorID, int ID) {
        this.id = ID;
        this.message = message;
        this.authorID = authorID;
    }
    int commentCount = 0;
    int endorseCount = 0;

    @Override
    public int getCommentCount() {
        return commentCount;
    }

    @Override
    public int getEndorseCount() {
        return endorseCount;
    }

    @Override
    public void incrementCommentCount() {
        commentCount++;
    }

    @Override
    public void incrementEndorseCount() {
        endorseCount++;
    }

    @Override
    public void setCounts(int commentCount, int endorseCount) {
        this.commentCount = commentCount;
        this.endorseCount = endorseCount;
    }

    /**
     * The method returns the contents of the class as a string value. Used in
```

```
56 this format for saving and loading the
57      * platform.
58      * @return String containing object's contents in the format "Class,id,
   message,authorID,commentCount,endorseCount".
59      */
60     @Override
61     public String toString() {
62         return "Post," + id + "," + message + ',' + authorID + "," +
   commentCount + "," + endorseCount;
63     }
64 }
```

```java
package socialmedia.socialmedia;

/**
 * BasePost is an <i>abstract class</i> used to construct the classes of <strong>Post</strong>, <strong>Comment</strong> and
 * <strong>Endorsement</strong>.
 * <p>
 * Its existence serves to allow for all components to be stored in one homogenous list, while also allowing Endorsement
 * to inherit Post's methods without inheriting the Interactable interface.
 *
 * @author Daniel Casley
 * @version 1.0
 */
public abstract class BasePost {

    static int nextID = 0;

    /**
     * The method resets the nextID counter stored in the BasePost abstract class. This means that the next BasePost of
     * any type will have the ID of 0.
     */
    public static void resetCounter() {
        nextID = 0;
    }

    /**
     * The method returns the nextID counter stored in the BasePost abstract class.
     * @return The nextID counter
     */
    public static int getCounter() {
        return nextID;
    }

    /**
     * The method sets the nextID counter to a specified number.
     * @param counter Number to set counter to
     */
    public static void setCounter(int counter){
        nextID = counter;
    }

    protected int id;
    protected String message;
    protected int authorID;

    /**
     * The method returns the message stored in the post.
     * @return Message stored.
     */
    public String getMessage() {
        return message;
    }

    /**
     * The method returns the ID of the post.
     * @return ID stored.
```

```
56      */
57     public int getId() {
58         return id;
59     }
60 
61     /**
62      * The method returns the ID of the user that created the post.
63      * @return Author ID stored.
64      */
65     public int getAuthorID() {
66         return authorID;
67     }
68 
69     /**
70      * The method sets the authorID to a new number.
71      * @param newID Number to set authorID to.
72      */
73     public void setAuthorID(int newID) { authorID = newID; }
74 
75     @Override
76     public String toString() {
77         return "";
78     }
79 }
80 
```

```java
package socialmedia.socialmedia;

/**
 * Endorsement is a class that simply inherits BasePost, while adding capabilities for an original post ID, allowing us
 * to use filter operations to relate it to an original Post or Comment.
 *
 * @author Daniel Casley, Benjamin Richmond
 * @version 1.0
 */
public class Endorsement extends BasePost {
    int originalPostId;

    public Endorsement(String message, int authorID, int originalPostId) {
        this.id = nextID++;
        this.message = message;
        this.authorID = authorID;
        this.originalPostId = originalPostId;
    }
    public Endorsement(String message, int authorID, int originalPostId, int id) {
        this.id = id;
        this.message = message;
        this.authorID = authorID;
        this.originalPostId = originalPostId;
    }

    /**
     * The method returns the ID of the original post that was endorsed.
     * @return ID of the endorsed post.
     */
    public int getOriginalPostID() {
        return originalPostId;
    }

    /**
     * The method returns the contents of the class as a string value. Used in this format for saving and loading the
     * platform.
     * @return String containing object's contents in the format "Class,id,message,authorID,originalPostId".
     */
    @Override
    public String toString() {
        return "Endorsement," + id + "," + message + ',' + authorID + "," + originalPostId;
    }
}
```

```java
package socialmedia.socialmedia;

import socialmedia.socialmedia.interfaces.Interactable;

/**
 * Comment is a class that inherits the Endorsement class (the capabilities to store an <strong>original ID</strong>),
 * as well as implementing the Interactable interface, allowing it to be commented and endorsed.
 *
 * @author Daniel Casley, Benjamin Richmond
 * @version 1.0
 */
public class Comment extends Endorsement implements Interactable {

    public Comment(String message, int authorID, int originalPostID) {
        super(message, authorID, originalPostID);
    }
    public Comment(String message, int authorID, int originalPostID, int id) {
        super(message, authorID, originalPostID);
        this.id = id;
    }

    int commentCount = 0;
    int endorseCount = 0;

    @Override
    public int getCommentCount() {
        return commentCount;
    }

    @Override
    public int getEndorseCount() {
        return endorseCount;
    }

    @Override
    public void incrementCommentCount() {
        commentCount++;
    }

    @Override
    public void incrementEndorseCount() {
        endorseCount++;
    }

    @Override
    public void setCounts(int commentCount, int endorseCount) {
        this.commentCount = commentCount;
        this.endorseCount = endorseCount;
    }

    /**
     * The method returns the contents of the class as a string value. Used in this format for saving and loading the
     * platform.
     * @return String containing object's contents in the format "Class,id,message,authorID,originalPostId,commentCount,endorseCount".
     */
```

```
56     @Override
57     public String toString() {
58         return "Comment," + id + "," + message + ',' + authorID + "," +
   originalPostId + "," + commentCount + "," + endorseCount;
59     }
60 }
```

```java
package socialmedia.socialmedia.interfaces;

/**
 * Interactable is an interface that sets the groundwork for the functionality to receive Comments and Endorsements.
 * <p>
 * Its primary functionality is allowing lists to be made consisting only of Interactable posts, and for aiding in
 * throwing NotActionablePost exceptions.
 *
 * @author Daniel Casley, Benjamin Richmond
 * @version 1.0
 */
public interface Interactable {
    /**
     * The method returns the endorsement count of the post.
     * @return The comment count of the post.
     */
    int getCommentCount();

    /**
     * The method returns the endorsement count of the post.
     * @return The endorsement count of the post.
     */
    int getEndorseCount();

    /**
     * The method sets the comment and endorse counts to two specified numbers.
     * @param commentCount Number to set comment counter to
     * @param endorseCount Number to set endorse counter to
     */
    void setCounts(int commentCount, int endorseCount);

    /**
     * The method increments the comment count by one. Used when a post is created.
     */
    void incrementCommentCount();

    /**
     * The method increments the endorsement count by one. Used when a post is created.
     */
    void incrementEndorseCount();
}
```